NTNU

Norwegian University of
Science and Technology

# A mobile extensible architecture for implementing ubiquitous discovery gestures based on object tagging

Simone Mora

Master of Science in Computer Science
Submission date: July 2009
Supervisor: Babak Farshchian, IDI

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

# Abstract

Mobility of people and their interactions with devices and services that every day become more pervasive in our life is a valuable challenge for system engineers. Locate friends, retrieve multimedia informations from physicals objects, have medical assistance remotely is going to be a commodity for a more and more wide part of the population, including elderly people. In this scenario have an easy way for discovering and communicating with third party services and resources that we encounter in our every day life is going to make the difference between an enjoyable user experience or a frustrating one that quickly leads to the abandon of a system.

Building on the work done in the past about resources discovery and management I study a solution for user-centered interactions with resources and services dynamically discovered and used by the user in nomadic environments. The solution designed make use of embedded devices, addressing the problem encountered in the research from both end-user and developer point of views.

The solution proposal make a full use of the service oriented architecture (SOA) concepts focusing on the goal of achieving the most natural human interaction with devices that the user discover on his way, keeping at the same time the framework architecture lightweight and easily extendible by third-party developers, as the SOA paradigm requires. Accessibility and extensibility are achieved on the end-user side by deploying software needed for the personal device (UbiNode) on most common smartphones and providing a easily understable Graphical User Interface; on the developer side by creating a pluggable framework based on xml and Eclipse eRCP runtimes for a fast development of multiple user interface that fits the constrains of the device in which are them deployed in.

Work done consists in design and implementation of several platform components and development of prototypes that takes profit from the overall architecture. Developed modules have been deployed end tested on handheld devices.

UbiCollab provides a solution platform for ubiquitous collaboration scenarios and this thesis has been carried out as a contribution to it.

**Keywords:** Ubiquitous Computing, User Centered Collaboration, User Interfaces, Mobile Devices, Discovery Gestures, Object Tagging, UbiCollab.

# Preface

This thesis is submitted as the final work for the degree of Master of Science in Computer Science that has been taken by the writer at the Norwegian University of Science and Technology NTNU (periods Jan-Jun '07 and '09), and at University of Bergamo, Italy. The report is based on the research work conducted by the writer from January 2009 throughout November 2009 on a project assignment given by the Department of Computer and Information Science (IDI). The work performed is a contribution to the UbiCollab platform. UbiCollab is a technological platform for supporting mobile and ubiquitous collaboration.

In this work is developed an architecture for building user interfaces for core components and applications in UbiCollab. This work builds and takes ideas on the work that Kim-Steve Johansen did in 2007 about resources discovering. The report presents the design, implementation and evaluation of a user interface framework, prototypes of applications which take advantages from the UI Framework.

The general task description for this work is included in Appendix A.

I wish to thank my supervisor at NTNU Babak Farshchian and co-adviser Monica Divitini for the excellent support and valuable feedbacks. Conversations and group discussions with them and other students have been extremely interesting and motivating and have played a needful role in achieving project goals.
I'm also grateful to prof. Stefano Paraboschi for the support to my work from Italy and for the help, suggestions and motivations I got from him during the years.

Trondheim, November 20, 2009

Simone Mora

# Contents

# Chapter 1

# Introduction

Separating computer system in Central Processing Units and peripheral devices has been the first step made in computer modularization and and service approach. The first computer I owned was a Commodore 64, it was 1990 and the era of home computing was just started. The C64 followed the Commodore PET (Personal Electronic Transactor) in the 80s. As the name suggests the PET was the first all-in-one home computer, composed of a CPU, a QWERTY keyboard, a monochrome monitor and a data tape unit, all framed together in a tough metal case for the considerable weight of about 20kg. The C64 was built following a total opposite design pattern: just the keyboard was wrapped with the CPU, in addition it came with interfaces for connecting external resources like data tapes, printers and game pads sold as optional devices; moreover I think that the killing feature that convinced my parents to buy it for me, was that they hadn't to buy me even a computer screen (really expensive at that time) but they could just connect it to the standard TV we were sharing. This design approach (combined with a good marketing strategy) let the C64 became the best-selling personal computer of all the time.

Nowadays we live in the ubiquitous computing era and we have to deal day-to-day with complex modular systems both for work and leisure. In these

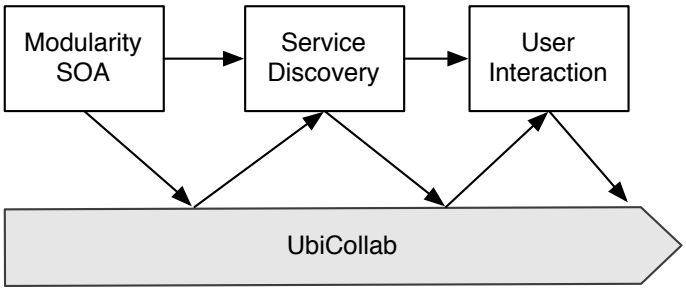



**Figure 1.1:** Knowledge transfer in UbiCollab

systems a module can be small enough to be embedded in a ordinary object like a book, a key or a clothes, therefore technologies for discovering and connecting these resources has become a cornerstone topics for ubiquitous computing. As matter of fact, connected devices and objects in our physical environment necessitate mechanisms for finding these objects before being able to use them. Resource discovery (also called service discovery) is the common term used to denote technologies that assist us in finding/discovering available services/resources/objects around us. On top of that, any discovery operation that is user initiated involves the presence of one or more HMI - Human Machine Interaction mechanism. A friendly User Interface in Service Discovery is hard to achieve since it has to hide the underlying complexity of the service oriented architecture from the end user perspective (Fig. 1.1).


**Figure 1.2:** Commodore PET and Commodore 64

The following scenario could clarify these concepts:

*Arne is a old retired man with chronic hearth diseases. On his 70th birthday his sons, Bjrn and Silje, gave him a UbiNode smarthphone and a Home Automation kit to control electrical devices such as lights and heating systems directly from his bed.*
*Installing the kit in Arne's house they showed him that is possible to add devices controlled by the smarthphone typing a numeric code, reading a barcode with the phone camera or reading a RFID tag with a pen reader provided with the UbiNode. Arne started to use the UbiNode to turn off his rooms lights from his bed before falling asleep and turn them on in the morning.*

*Arne, because of his hearth diseases, needs frequent check-up with his cardiologist, dr. Tor. Arne lives far from the hospital and have to be driven by his sons every time he needs a check-up. For this reason the hospital sent him a biomedical shirt which checks earth pulse and blood pressure data. These bio-shirts are connected via WiFi with a smartphone and the patient is able to check heart's health by a green-yellow-red status light rendered on the phone screen; moreover the smartphone can inform dr. Tor via SMS in case of hearth attack.*
*Arne once received the shirt, clasps his UbiNode and read the RFID tag on the shirt. The shirt got activated and the UbiNode starts to show feedbacks collected by it.*

The UbiCollab project aims to cover these research fields and this thesis is provided as support to it. We elaborated a solution proposal for User Interactions in User Centered Service Discovery scenarios, modules and proof-of-concepts applications have been implemented.

The UbiCollab platform is also getting in touch with other research projects, such as Awareness Services and System - Towards theory and ReAlization (ASTRA), an ongoing project where among others NTNU is a participant.

**The rest of this chapter is organized as follows:**

In Section 1.1 the motivation for this work along with goals and contributions is described. This will give an overview of what is being accomplished by this work and how this fit into the larger UbiCollab project. Research problems are also pointed out.

In section 1.2 research method which has been followed to achieve project goals is described.

In section 1.3 we will introduce main concepts and context related topics involved in UbiCollab.

In section 1.4 the main Ubicollab concepts and terminologies will be depicted.

Finally, Section 1.5 will be described how the rest of the report is organized.

## 1.1 Motivation and Contributions

### 1.1.1 Motivation

The main motivation beyond this work is to do researches in Resource Discovery and Management fields, develop solutions and implement proof-of-concepts applications in UbiCollab.
Since Ubicollab Resource Discovery subsystem aspire to be *user-centered*, our research focus will not cover just modules logic implementation but also we take in account users, their interactions with the system and their needs; because it doesn't matter how much a system could be efficient, reliable, advanced: if a person without any technical knowledge on ubiquitous computing feels frustrated using it or hardly adaptable to his needs, we have failed and all the unseen work on the bottom of the architecture is mostly useless. Anyway before refine the friendliness of user interactions we need to have a reliable solution on our system fundamentals and especially this solu-

tion must be deployed on handheld devices. The achieving of this goal implies to deal with a lot of different third-party components and obtaining a stable configuration, connecting different technologies not born to talk together could also be considered a valuable result even if it is a prerequisite to build a proprietary system over. Moreover, designing a system like UbiCollab which aims to support collaboration among users in a wide arena of scenarios, from simple personal devices management, to forefront healthcare applications; involves the design of a system that has to be really adaptable to different problem domains, loosed coupled and platform independent.
These are the main guidelines that aimed my work: design a highly efficient loosed coupled architecture keeping on sight that the benefits from my work has to been available and usable even by elderly and children.

### 1.1.2 Contributions

In this project an extensible architecture to handle different User Interactions in UbiCollab is proposed, implemented and evaluated. The solution proposed is focused on interaction in the Resource Discovery area but thanks to the solution modularity it can be adopted to provide User Interactions support even to the other UbiCollab modules such as the identity manager or the space manager.

This work presents the following contributions:

*Theoretical works:*

- Research on User Centered Service Discovery protocols and Object Tagging: a research about user-centered service discovery protocols and technologies used to tag resources that have to be discovered.
- Research, comparison and evaluation of different User Interaction technologies: an analysis of possible user interactions according to the direct-manipulation paradigm.

- Research, comparison and evaluation of different Java Virtual Machines for handheld devices: a comparison chart among possible JVM implementations compatibles with our platform.
- Research, comparison and evaluation of technologies for Graphical User Interfaces development: a comparison among tools and technologies available for GUI implementations.
- Elaboration of a test Scenario for platform evaluation purpose.

*Engineering works:*

- Update of the UbiCollab modules developed in previous works and standardization of the module unit.

  Design and implementation of the following software components:

- UbiCollab eWorkbench: the user interactions manager
- UIToolkit: an SDK for developing UbiCollab Applications without extensive coding.
- AstraConnector: sharing and integrating services with the ASTRA Project.
- Type-a-Number Service Discovery Plugin.
- RFID Service Discovery Plugin.
- 2D Barcodes Discovery Plugin.
- ImageViewer App.
- SharedScreen Proxy.
- SharedScreen WebService for Tablet PCs.
- SharedScreen WebService for IDI OpenWall.
- Setup of the platform on mobile devices, testing, user testing and benchmarks.

### 1.1.3 Research Problems

Because of this project have also to deal with not strictly technical topics like user behavior and user friendliness is not always possible to achieve an user friendly interface and a lightweight service oriented architecture at the same time. This trade-off have brought me to decide to focus more on the architectural/technical domain of the problem, in order to bring to the platform multiple user interactions capability, instead of choosing to support one particular UI approach and implement it. This decision is justified by the central idea of UbiCollab which is to support collaboration in mobile, nomadic environments, where neither environmental variables nor user groups are not prior defined. It means, for instance, that a fast gesture-based interaction with the system could be the preferred choice for a youth traveling on a bus whereas a voice-based one with lights notifications can be the most preferred by an elderly person sat on his wheelchair. For this reasons our efforts are most driven to provide a framework which supports all this interaction mechanisms end let the user choose which one is the best for a given context exploiting the environment. Moreover since UbiCollab Applications has to be written by developers without extensive coding [1] we try to provide this multiple UI support in the most developer-friendly way.

## 1.2 Research Method

The research method I adopted in my work time is schematically presented in figure 1.3. All the steps were reviewed and corrected by feedbacks from tests and meetings with my supervisor professor Babak Farshchian and co-advisor professor Monica Divitini.

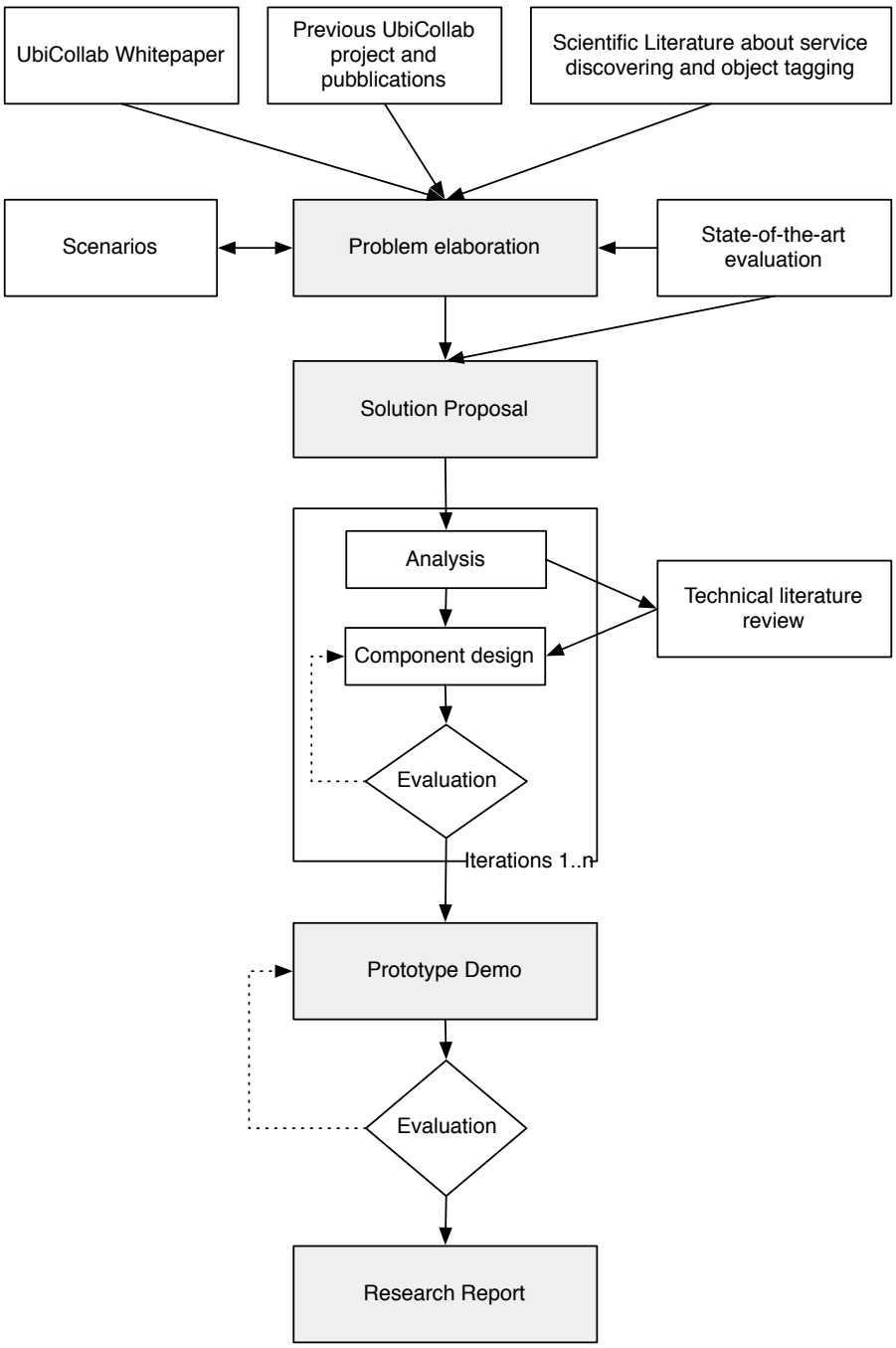



**Figure 1.3:** Research Methodology

## 1.3 UbiCollab Context and backgrounds

This section will introduce the main concepts and context related topics involved in UbiCollab. For a more complete description of the platform, refer to the UbiCollab White Paper [1].

### 1.3.1 Ubiquitous Computing

The contemporary writer William Gibson said in an interview :

> *"One of the things our grandchild will find quaintest about us is that we distinguish the digital from the real"*

this could be considered a manifesto of Virtual Reality (VR) and even a source of concerns and discussions for psychologists and philosophers. Ubiquitous computing claims to be the opposite of Virtual Reality [2], it brings back the informations to physical objects and these informations become accessible through a natural interaction with them. The focal point has to move from machines to people and people's needs; moreover the computation has to be thoroughly integrated into everyday objects and activities that should become invisible. The associated "many computer per person" concept is considered the third wave of computing, coming after the first wave "Many people per computer" and the second w. "A person per computer".

The idea of Ubiquitous Computing (or Pervasive Computing) was first formally introduced as research field by Mark Weiser's seminal paper in 1991 [3] and is now explored by a number of leading technological organization, such Xerox's Palo Alto Research Center (PARC), IBM, The Massachusetts Institute of Technology (MIT).

### 1.3.2 Computer Supported Collaborative Work (CSCW)

CSCW is an interdisciplinary field of research that is concerned about how people work or collaborate together, and how technology can be used to support collaborative activities and coordination. The objective is thus to create a basis for designing computer systems by establishing the nature and requirements of collaboration among people [4]. UbiCollab aims to support CSCW but even looks to support a wider domain of applications including leisure activities and personal healthcare.

## 1.4 UbiCollab

UbiCollab (short for Ubiquitous Collaboration) is a service platform for provision of basic services for supporting collaboration among people. UbiCollab make extensive use of earlier CSCW research, and extends this research with insights from ubiquitous computing and wireless services [5]. It provides a platform that captures the commonalty of collaborative applications and provides generic mechanism for applications to be built without extensive coding, in order to naturally support collaboration in any situation the user are in [1].
UbiCollab tries to be domain-independent and providing only the basic functionality, is therefore following an open innovation approach where third party applications play an equally central role as the platform itself. Integration with physical environment where collaboration happens is a key aspect of UbiCollab.

UbiCollab architecture follows the Service-Oriented Architecture (SOA) approach. UC is implemented as a collection of independent components in form of dynamically deployable services that can be deployed and used independently on a mobile device.
Each UC component is being developed to cover a very specific area of re-

sponsibility. Components can be mixed and used together in different configurations (compositions) decided by the scenario using them. Only those components that are needed by a specific user (and his/her applications) will be deployed on his/her mobile device [1].

UbiCollab is an OpenSource project registered with Sourceforge[1], and the source code is available under the Apache License v2.0.
Everyone, users and developer, can try it and contribute to the growing of UC developing applications for it or just sending feedbacks.

### 1.4.1 The Human Grid

The abstract concept of a human grid constitutes the vision underlying UbiCollab. A human grid is a collection of people and their artifacts/resources connected together using UC platform technology, as schematically presented in figure 1.4 . Interactions in a human grid are supported using resources, artifacts, services, etc. imported into the grid by its participants. UC assists its users in building a human grid and supports communications among them, and they can be distributed geographically.

Human grid is adaptive and reconfigurable in that it will change its configuration in order to best fit context, services and artifacts that users have available in any given space. It may change its configuration and deployment configuration in order to assists users in a lot of different scenarios, from work-collaborative related to health care assistance, as the user moves from a space to another.

### 1.4.2 UbiNode

Each user in UC is represented and assisted by a mobile device called a UbiNode.

[1] Available at: http://ubicollab.sourceforge.net/

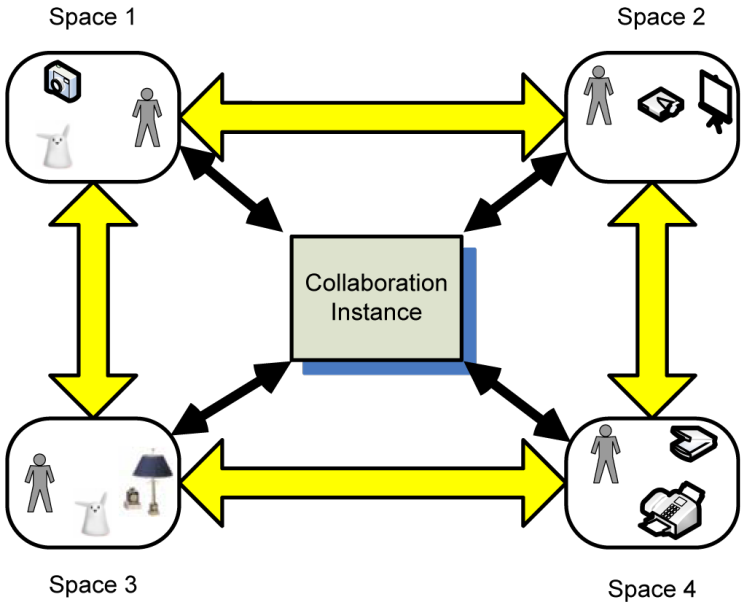



**Figure 1.4:** The Collaboration Grid

UbiNode is a network-enabled device that acts as a personal server, running a subset of the main UC components and some of user's applications designed for it. This means that each user has his/her own instance of a Resource Discovery Manager, Service Domain Manager, Space Manager, CI Manager etc. running locally on his/her UbiNode.

As reported in figure 1.5, UbiNode is organized in a "platform space" where core components resides and a "user space" where each user can store and run his/her application which communicate with external devices. All the components allows interaction with other applications exposing WebService interfaces.

Complete independence among UC components allows us to outsource all composition tasks to the applications and guarantees a high level of modularity in the architecture of a UbiNode, in accordance with the SOA approach.

Currently the UbiNode is deployed on Windows Mobile smartphones and tested on devices reported in Appendix C.

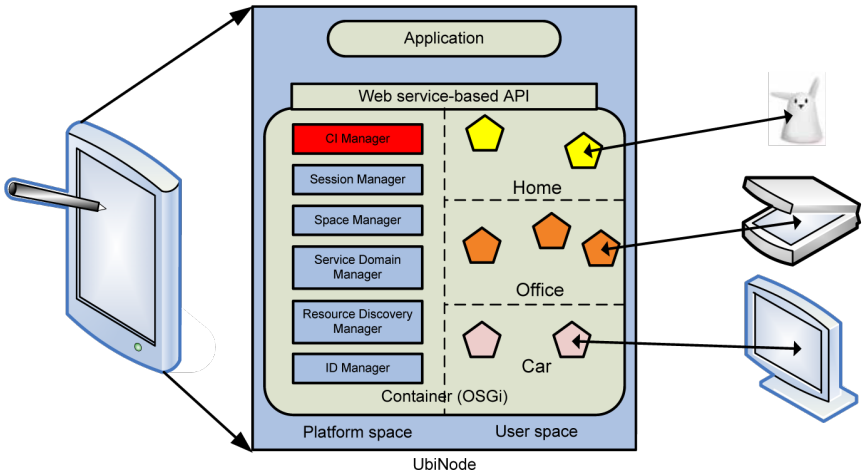



**Figure 1.5:** The UbiNode

### 1.4.3 Services, Service Proxies and Service Domain

UbiCollab allows a group of distributed users share an arbitrary set of information, and be aware of each other's physical location. These two concepts support the idea of mobility in distributed online collaboration by allowing users be aware of each other's location. A step further in supporting natural and ubiquitous collaboration is implemented by the UC concept of Services, Service Proxies (SPs) and Service Domain (SD). The goal is to allow users deploy external resource in their collaboration with others. This will allow for a natural way of collaboration by for instance using dedicated devices and services in a meeting (such as projectors, whiteboards) in an unknown environment. For instance, a table lamp is used to signal the availability of a contact in a instant messaging application developed for UC. A nabaztag rabbit is used for the same purpose, while a digital camera is used to take photos and share it with the other users. A GPS-enabled clock is in addition used to provide the user's current GPS coordinates. The notion of a Service is used in UC to denote such external resources brought into a UC environment in order to be used in collaboration. The mechanism used to connect to these Services (which might be devices, web services etc.) is through a dedicated

Service Proxy (SP). SPs are discovered using UC's service discovery mechanisms (described later) by e.g. reading an RFID tag or a Barcode attached to the actual Service. This tag refers to a Service Advertisement used in order to dynamically locate, install and set up an SP at user's wish. In order to facilitate the management of many SPs that a user potentially might have, each user is assigned a Service Domain (SD). All the SPs installed by a user are registered and maintained by that user's SD, which is also responsible for other tasks such as secure access to SPs and protection of user's privacy. Each SP is in addition tagged using a Space identifier. This means that UC can support location-aware access to Services. For instance, if a user resides in a Space called "My home", only Services labeled with "My home" might be available to that user's applications by default. Since the main purpose of UC is to support ubiquitous collaboration, we need to allow users share their Services with other users. This is done through a process we call service publishing. A user can choose to publish one of his/her Services in a CI.

### 1.4.4 Resource Discovery

Resource Discovery (RD) Manager implements a mechanisms for accessing and integrating external resources in UC. Discovery of resources can be done in many different ways, each modality is characterized by a different user interactions. In UC we wanted to avoid creating yet another discovery protocol. RD Manager uses so-called discovery plug-ins to enable interaction with our native discovery mechanisms. For instance, a plug-in can use a camera embedded in the UbiNode to take a photo and decode a 2D barcode which encode an URL. All plug-ins thus return to the resource discovery module, a URL which points to the Proxy Service (an OSGi bundle) for the discovered resource. This URL can be passed to Service Domain Manager, which will use it to install the Proxy Service in user's Service Domain. When a resource is correctly installed it become a service that exposes a public interface and thus can by used by applications. The overall procedure is illustrated in

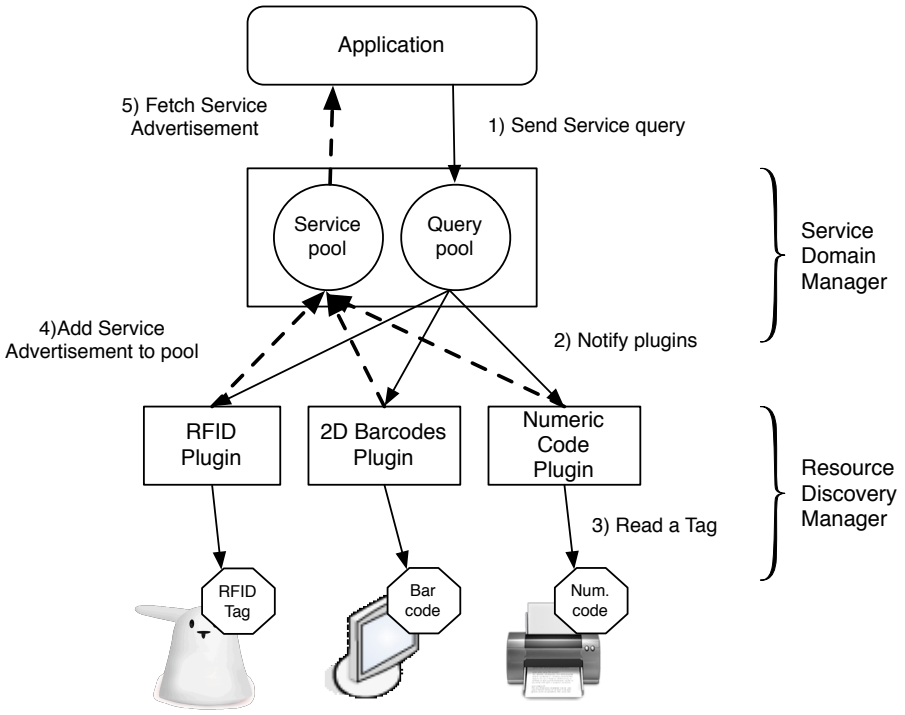



**Figure 1.6:** The Resource Domain Subsystem

figure 1.6.

My work builds on and take advantages from APIs provided by the Resource Discovery subsystem that Kim-Steve Johansen has developed [2], which provides an indispensable technical background for an User Centered interaction with the system.

## 1.5 Report Outline

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

**Chapter 2** investigates general concepts related to User Interaction with the resources and evaluates which set of interactions are more suitable for Resource Discovery operations thus, defining the concept of *Discovery Ges-*

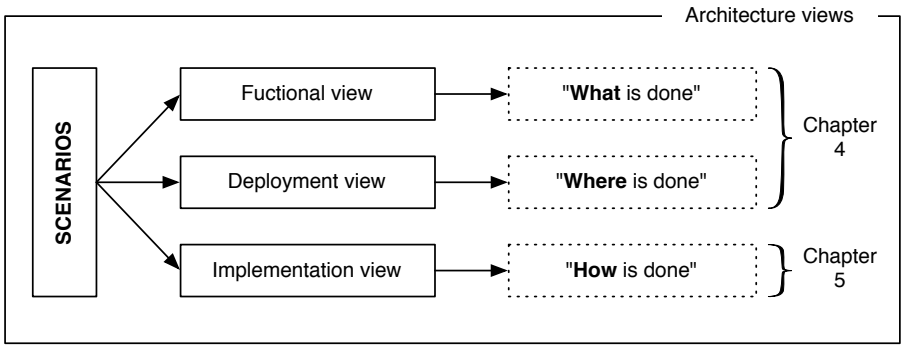



**Figure 1.7:** 4+1 View model adopted in this report

*tures* a personalized, user centered solution for handling Resource Discovery in UbiCollab is defined.

**Chapter 3** presents the solution proposal for the main research problem. It will be described from the functional and deployment point of view, in accordance with the "4+1 Architectural Model View" technique [6] (figure 1.7). This chapter will give an high-level description of concepts and a critical explanation of the choices made in the components design work. It also shows mockups and real deployment examples of GUIs and other components developed as part of the research.

**Chapter 4** reveals how the proposed solution has been implemented. It exposes keypoints and inner functionality of the platform components and also schemas and diagrams that have been derived.

**Chapter 5** describes how the solution has been tested and evaluated. The evaluation are based on a prototype developed by the solution proposal and take advantage of feedbacks collected during the public workshop occurred on April, 30th.

**Chapter 6** concludes the report by presenting results and contributions, giving an overall evaluation of the research process and the report itself. Finally, some ideas and thoughts for future work to the UbiCollab platform will be suggested.

**Appendices:**

- Appendix A contains the problem assignment that was given and related scenarios.
- Appendix B presents technical informations about UbiCollab current distributions, install procedures and versioning.
- Appendix C contains the hardware specifications of devices used in the research and during tests and evaluations.

# Chapter 2

# Problem Elaboration and Analysis

## 2.1 Problem Definition

### 2.1.1 Approaches to User Interactions for Resource Discovery

Nowadays a lot of resource discovery systems have been developed, some of them are designed for wired and wireless network such as Jini, uPnP, Apple Bonjour, and others just for wireless, like the Bluetooth advertising system. These systems can also include a proprietary communication protocol among devices, as Bluetooth does, or can work over standards protocols like ethernet or wifi LANs as uPNP and Bonjour do.
Therefore a valuable question is: why we need to develop a new resources discovery system instead of using a standard one? Recalling that UbiCollab targets nomadic scenarios and is deployed on handled devices, there's two main arguments that justify efforts on designing a new RD approach:

1. standards RD protocols like uPNP even if are really efficiency in dis-

covery and network operations are too generic and lacks in effectiveness: how many time, for instance, you searched for a printer in your workplace on your PC and the system returns a long list of printers, including ones that maybe resides in another building or are locked in a colleague office? If you have experience about the place where you are, consequently you know the resources location and you can choose the closest to your position, but how a guest who enters in your workplace for leading a conference can choose the closest printer?

2. Handheld devices have limited resources and user interfaces, thus users should be presented an optimized short list of available resources instead of a long list of discovery results .

3. Improvements introduced by self advertising system, as the Bluetooth one, are still not effective enough to address problem mentioned in points 1 and 2. Procedures for coupling devices in the BT domain is sometimes tricky and not user-friendly enough for some parts of the population, including elderly.

Searching for the most appropriate resource we should exploit the environment in which we are and interact with objects that show us a *service advertisement*. According with the Ubiquitous Computing ideas we want to bring the information back to the objects and interact with them in a natural way. That's the reason because for UbiCollab we chose to develop a resource discovery system based on ***Discovery Gestures***, instead of a service listing search approach.

A *Gesture* is any physical movement that a digital system can sense and respond to without the aid of a traditional pointing device such as a mouse or a stylus [7] A *Discovery Gesture (DG)* is a gesture settled for resource discovering, it takes benefits and grow up from study in the Automatic Identification and Data Capture (AIDC) fields mixed with Service Discovery Protocols.

A discovery gesture can be pointing to a RFID tag, taking a photo of a Barcode, typing or saying a code. We support both gestures and discovery gestures. Gestures are used for a natural interaction with applications: browsing photos sliding fingers on the screen, for instance; these can be implemented even by third-party applications. Discovery gestures rather are a set of predefined gestures developed by UbiCollab crew and used in resource discovery operations, these are considered the most natural way to interact with resources, because the use of pointing provides a natural way of communicating [2]. A single Discovery Gesture is implemented in a plugin for the UC Resource Discovery architecture.

The goal of this approach should be to switch from a retroactive resource discovery to a proactive RD (figure 2.1) where the user has a full control on the discovery process.

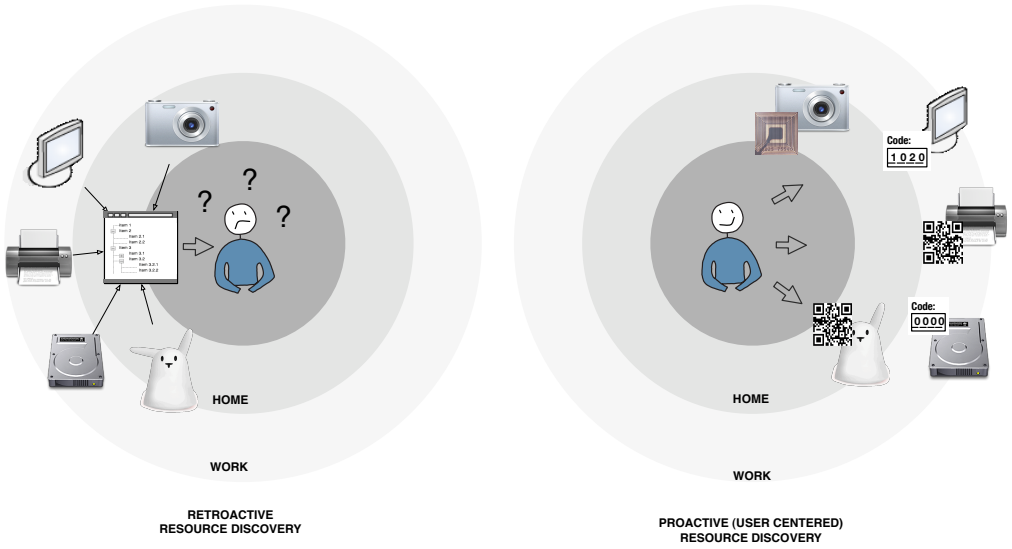



**Figure 2.1:** Retroactive vs. Proactive Resource Discovery

*This kind of approach will contribute to the development of a "Personalized Resource Discovery" where services primarily advertised to the user are the ones closer to his location and those that fits user's interest.*

User Interactions in UbiCollab are not just referred to resource discovery, these are core tasks and their UIs are designed by the internal developers, but as working with an open source project we aim to support new developers and provide them the easiest way to add their own contribute.

### 2.1.2 Resource Discovery Actions and Gestures

At the present time UbiCollab works towards four Discovery Gestures:

1. Point an RFID tag
2. Type a number
3. Take a photo of a 2D Barcode
4. Say a number

The Discovery Gesture 1 has been first investigated by Kim-Steve Johansen in his master thesis [2]. The Discovery Gesture 2-3-4 has been developed during my researches, its implementation will be reported in Chapter 4. The "Say a number" discovery gestures has been investigated and will be implemented in a future work. An UbiCollab distribution can include all these discovery gestures and the user can choose which one to use according with his/her preferences and environmental factors.

## 2.2 Approaches to User Interaction

### 2.2.1 Touchscreens vs Free-form Interaction

Currently, most gestural interface can be categorized as touchscreens or free-forms. Touchscreen gestural interfaces (also called touch user interfaces-TUIs) require the user to touch the device directly. This puts a constraint on the types of gestures that can be used to control it. Free-form gestural

interfaces don't require the user to touch or handle them directly. Sometimes a controller or a glove is used as an input device, but even more often the body is the only input device for free-form gestural interfaces

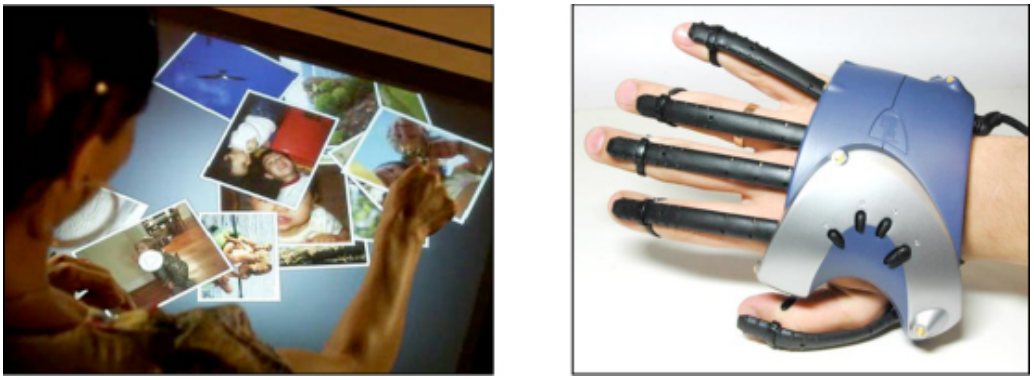


**Figure 2.2:** Microsoft Surface (a forefront touchscreen) and Essential Reality P5 Glove (the first commercial controller for gestural interfaces)

### Touchscreens

The first concept related to touchscreens usability be born in a seminal 1983 paper [8] written by the Maryland professor Ben Shneiderman which forged the concept of direct manipulation. Direct Manipulation is the ability to manipulate digital objects on a screen without the use of command-line commands acting, for example, dragging a file to a trash on your desktop instead of typing *del* into a command line.
Shneiderman was mostly talking about mice, joysticks, and other input devices, since at that time (1983) they where considered innovations connected with the growing desktop metaphor.
Touchscreens and gestural interfaces take direct manipulation to another level. Touchscreen users can simply touch items they want to manipulate right on the screen itself, tapping fingers on buttons, dragging icons, scrolling texts. These gestures are performed on a physical surface, thus are also called *tangible interfaces*. In the next section we will see the use of gesture where a tangible support is missing. This is the ultimate in direct manipulation:

using the body to control the digital (and sometimes even the physical) space around us.

**Free-form Gestural Interfaces**

Free-form gestures exploit the movement of body limbs in the air (fingers and arms movement, head rotations, etc) and map them to a set of computer commands. This body-computer information transfer is achieved by wearable devices which record physical variables like accelerations and orientation matching them against patterns and thus decoding the associated command. This represent a big improvement for User Interaction since the gesture are not just narrowed to a limited set that a touchscreen can support but are potentially infinite. Moreover the designed gesture can really reflect real-word operations like pointing a TV to turn it on (without the remote controller!) or shake a music player to listen a random song. As drawbacks, a successful free-form interaction is really hard to achieve since without the support of a physical surface to interact with is hard to return to the user feedbacks about the command interpreted by the calculator and this, together to the high complexity of the mathematical model that link physical variables to the gesture, can lower down precision and rate of success in understanding user willings.

### 2.2.2 The next step: Brain-Computer Interfaces

The next step in Human Computer Interaction (HCI) will probably be the Brain-Computer Interfaces. Since is demonstrated that electrical activity generated by ensembles of cortical neurons can be employed directly to control a digital device as a computer or a robotic manipulator, research on brain-machine interfaces (BMIs) has experienced an impressive growth. Today BMIs designed for both experimental and clinical studies can translate raw neuronal signals into motor commands that reproduce arm reaching and hand gasping movements in artificial actuators [9]. Clearly, these develop-

ments hold promises for the restoration of limb mobility in paralyzed subjects. However before this goal can be reached several bottlenecks have to be passed. These include designing a fully implantable biocompatible recording device, developing real-time computational algorithms, introducing a method for providing the brain with sensory feedback from the actuators, and designing and building artificial prostheses that can be controlled directly by brain-derived signals. By reaching these milestones, future BMIs will be able to drive and control revolutionary prostheses that feel and act like human arms. Anyway, despite the optimism raised by some new accomplishments, there are still many issues that preclude a widespread translation of experimental BMIs into practical applications. Indeed, most of the invasive BMIs have been tested only in experimental animals. Thus, despite recent enthusiasm much experimentation remains to be done before [9].

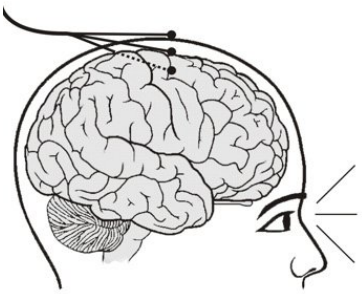


**Figure 2.3:** Brain-Machine Interface

## 2.3 Requirement Analysis

Requirement Analysis has been made in order to support elaboration and implementation of solutions. Note that the elaborated requirements don't replace those that have been formulated in previous works [2, 1, 10] rather they add requisites in the areas reported in figure 2.4, 2.5, 2.6, 2.7.

**Each requirement is tagged by IDs named RQ-RA-XY.**

RQ is the requirement category, can be:

- AR: Architectural Requirement
- FR: Functional Requirement
- NF: Non-Functional Requirement

RA represent the requirement area, can be:

- MB:Mobility Area
- UI: User Interaction Area
- DE: UI Design Area
- RD: Resource Discovery Plugin Design Area

XY is the number of the requirement

This classification will improve requirements traceability. Individual priority-assignment for each requirement have also been performed. Each requirement is weighted by **High (H), Medium (M) or Low (L)** importance, assessed for its priority in the component implementation schedule. In addition, a degree of difficulty (abbreviated as DoD) is given to better understand the strain and hence the time needed for implementing each requirement. This will be weighted the same way as priority, where high means "high difficulty" and low means "low difficulty".

| | ID | Description | Priority | DoD |
|---|---|---|---|---|
| ADAPTIBILITY | NF-MB-01 | System's component must run on handheld devices | H | M |
| | NF-MB-02 | System must at least run on a CDC Java Virtual Machine implementation | H | M |
| | NF-MB-03 | System's components can run on different OSGi implementations | M | M |
| | AR-MB-04 | System's components must have access to devices embedded resources (cameras, speakers, microphone) | H | H |
| | AR-MB-05 | System's components has to be optimized for mobile computation in order to save resources and improve mobile device autonomy | M | H |
| | AR-MB-06 | System's components size has to be compatible with PDAs limited storage memory | H | L |
| | AR-MB-07 | System's components must be backward compatibles with desktop PCs deployment | M | M |

**Figure 2.4:** Requirements in Mobility Area

| | ID | Description | Priority | DoD |
|---|---|---|---|---|
| ACCESSIBILITY | FR-UI-01 | User can choose a default Interaction mechanism | L | M |
| | AR-UI-02 | System's components must be able to publish multiple User Interaction mechanism | H | H |
| | NF-UI-03 | System must recognize UbiNode device features and constrains and consequently activate just the UIs available for the device | H | H |
| | AR-UI-04 | System's components must provide at least one GUI in case the device doesn't support more sophisticated UIs | H | M |
| | NF-UI-05 | System's UIs can use third-party libraries | M | M |
| | NF-UI-06 | System User Interaction Manager must provide UIs too | H | M |
| EXTENSIBILITY | AR-UI-07 | System's components just installed must show their UI mechanism without rebooting the platform or the device | H | M |
| | AR-UI-08 | System's UIs can be implemented without extensive coding | H | M |
| | AR-UI-09 | System's UIs can be implemented without knowing User Interaction Manager internal logic | H | M |
| | AR-UI-10 | System's UIs must bind themselves to the UI manager using a XML file | H | H |
| | AR-UI-11 | System's UIs must be embedded in the component which uses them | H | M |

**Figure 2.5:** Requirements in User Interaction Area

| | ID | Description | Priority | DoD |
|---|---|---|---|---|
| ACCESSIBILITY | AR-DE-01 | System's UIs must be designed following usability patterns | H | M |
| | AR-DE-02 | System's UIs provided with Service Discovery Plugin must advertise and illustrate the discovery gesture | M | L |
| | AR-DE-03 | System's GUI provided for User Interaction has to use OS native widgets | H | H |
| | AR-DE-04 | System's GUI must pass usability tests | M | H |

**Figure 2.6:** Requirements in User Interaction Design

| | ID | Description | Priority | DoD |
|---|---|---|---|---|
| EXTENS. | AR-RD-01 | RDPs must have a embedded service list | H | M |
| | AR-RD-02 | RDPs must update the resource list on startup | H | M |
| | AR-RD-03 | RDPs must provide an XML representation of the service list | M | M |
| ACCESSIBILITY | AR-RD-04 | RDPs must ask a confirmation to the user before installing the resource in the UbiNode | H | L |
| | AR-RD-05 | RDPs must provide a short description of the service discovered before installing it | H | M |
| | FR-RD-06 | User can discover RDPs through another RDP (RDPs are mutually discovered) | M | M |
| | FR-RD-07 | User can discover resources not on his\her sight from his\her viewpoint | M | M |
| | FR-RD-08 | User can discover resources far from his\her position | M | M |
| | FR-RD-09 | User can be an elderly or a child | H | H |
| | FR-RD-03 | RDPs must provide at least one UI mechanism | H | M |

**Figure 2.7:** Requirements in Resource Discovery Plugin Design Area

## 2.4 Discovery Gestures Comparison

According with what written about User Interactions we made a comparison chart of the three discovery gestures which have been designed. In each row are reported a feature and how much the Discovery Gesture accomplish that feature. Each DG is starred from one to four "+", one "+" means "low feature compliance", four "++++" means "high feature compliance".

| Feature | Discovery Gestures | | |
|---|---|---|---|
| | **Point an RFID tag** | **Type a number** | **Take a photo of a Barcode** |
| Resource advertisement can be hidden | ++++ (an RFID tag can be hidden in a plastic case) | + (Resource code has to be on sight) | + (Resource barcode has to be on sight) |
| Resource advertisement can be far from the user | ++ (an RFID tag as to be few meter far from the user) | ++++ (if the code label is big enough can be red from a quite long distance) | + (the barcode has to be few centimeters far from the device camera to be red) |
| Probability of discovering a wrong resource | +++ (if there are several RFID tags close each other wrong reads can happen) | ++ (the user can type a wrong code associated to another resource) | + (since the service advertisement, the barcode indeed, has to be seen in the camera viewfinder this DG is quite error-proof) |
| UbiNode Hardware Requirement | + (an RFID reader is still uncommon on most handheld devices) | ++++ (touchscreen or similar input mechanism are mandatory on PDAs) | ++ (Cameras are becoming a standard equipment on most PDAs, but the barcode decryption need even high computation power) |
| Ease of use for elderly and children | ++++ (point a resource is a natural way to interact) | +++ (Typing a number on touch screen nowadays is a quite popular interaction way with electronic devices) | ++ (Taking a photo of a barcode can be problematic and require to hold the device, thus the camera, firmly) |

**Figure 2.8:** Discovery Gestures Comparison Chart.

# Chapter 3

# Solution Proposal

This chapter will give a description of the proposed solution for User Interaction in UbiCollab, it present the design and overall functionality of the components.

An user interaction architecture based on a User Interaction Manager, the Ubicollab eWorkbench, has been designed and developed. By this component each developer can build an application for UbiCollab with a proprietary UI mechanism without extensive coding. Since a Service Oriented Architecture (SOA) is being used the importance of maintaining loose coupling between components is stressed in the solution. This is particularly evident in this component, which uses a pluggable solution.

In order to simplify the understanding of the overall system and help the development of future works a standardization of the UbiCollab components architecture and naming conventions has been made. A test scenario for the functionalities designed has also been elaborated.

Chapter start presenting GUI mockups driven by an Use Case analysis reported in Section 3.1.

In section 3.2 the User Interface Manager, called UbiCollab eWorkbench, and all the related concepts are presented. We make use of abstractions like

Views and Perspective in order to group concepts and highlight architecture keypoints.

In section 3.3 integration between UbiCollab an external framework is investigated.

Since each component can provide proprietary UIs but all of them has to furnish at least a GUI[1] in section 3.4 best practices and patterns for GUI design are investigated. These guidelines has been applied in the design of mockups showed in section 3.1.

In section 3.5 the overall platform structure is outlined, the inner component architecture is illustrated and technical names for the component parts are established.

In section 3.6 a scenario for platform testing is elaborated.

## 3.1 GUI Mockups

Consecutively you can find some mockups for Resource Discovery Plugins and Service Domain Manager GUIs; these are built on requirements study connected to the Use Case reported below.

***Goal: The User discovers and starts using a new resource.***

**Main Success Scenario:**

1. User starts the UbiCollab Interface on his smartphone
2. User choose the "Resource Management" perspective
3. User choose the "Discover new Resource" view
4. User choose a Resource Discovering Plugin
5. System show the Discovery GUI embedded with the Plugin

[1]Graphical User Interface

6. User perform the "Discovery Gesture" related with the plugin
7. System notify that a resource is found and show an information page about the resource
8. User read the resource description
9. User choose to install the resource on his UbiNode
10. System shows a progress bar to notify the progress status to the user
11. System confirm that the resource is installed and ready to be used by any UbiCollab Application

**Extension:**

7a: System cannot find the resource and show an error message
.1: User repeat the "Discovery action"

9a: User tap the discard option
.2: System returns at step 5

11a: System fails to install the resource and show an error message
.3: System returns at step 8

Mockups of GUIs related to the Use Case are reported in figure 3.1 and 3.2

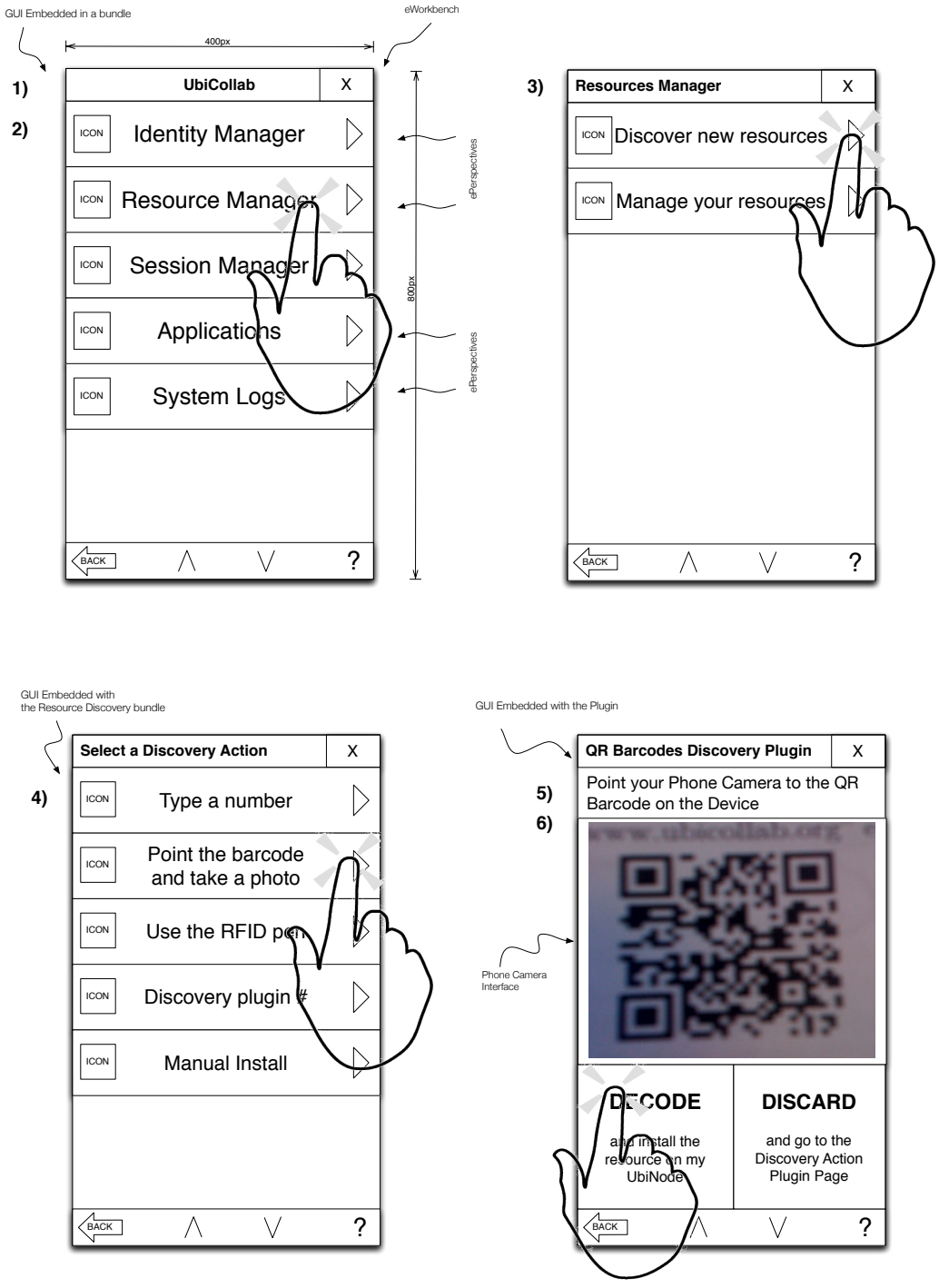



**Figure 3.1:** GUI Mockups, steps: 1,2,3,4,5,6

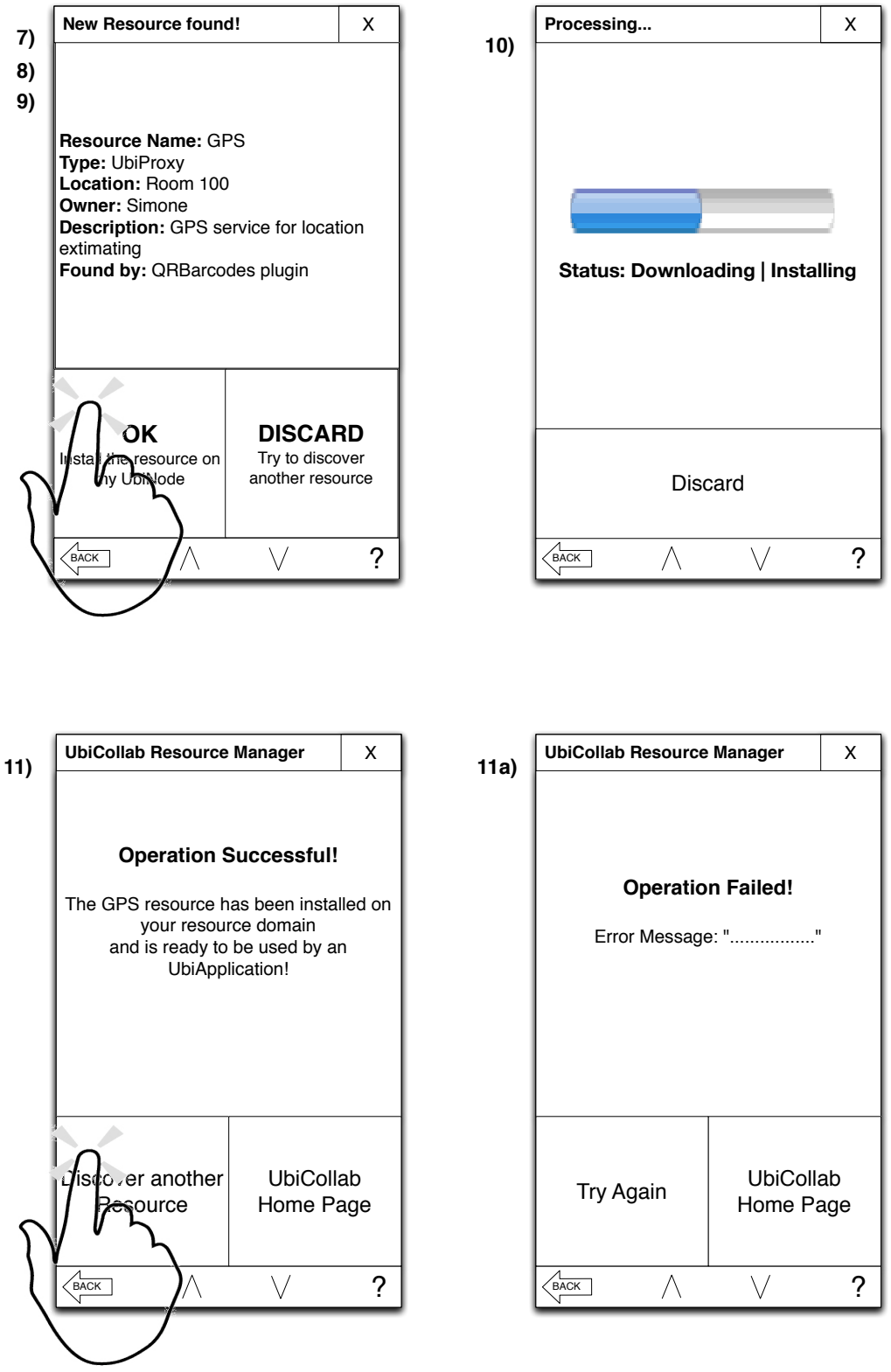



**Figure 3.2:** GUI Mockups, steps: 7,8,9,10,11,11a

## 3.2 User Interface Management in UbiCollab

In a high-level view we can consider each UbiCollab component whose provide User Interfaces as a combination of two layers working together: the *user abstraction layer* which connects the component to the UI Manager then consequently to the user and the *platform abstraction layer* which connect the bundle to other bundles and straight to the OSGi framework (figure 3.3).

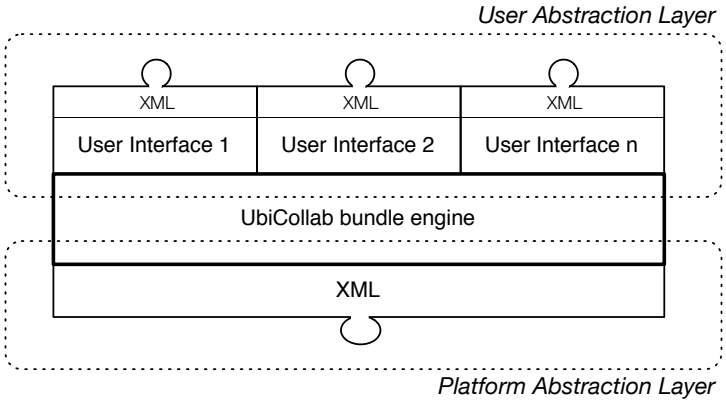



**Figure 3.3:** An UbiCollab Component

### 3.2.1 User Abstraction Layer

The User Abstraction Layer make use of three concepts which match users behavior: eWorkbench, View and Perspective (figure 3.4).

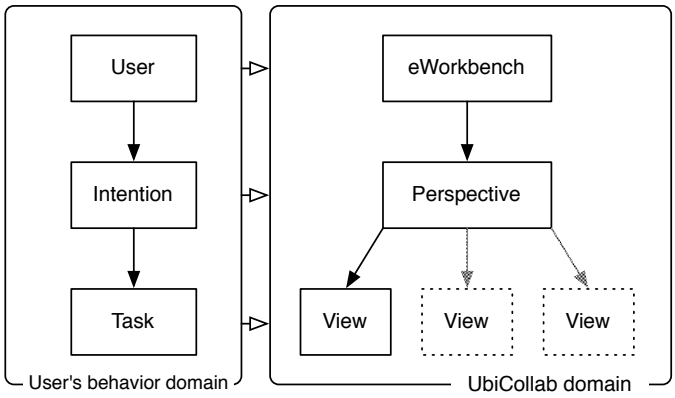



**Figure 3.4:** User's behavior matching in UbiCollab

### eWorkbench

The eWorkbench is the central component of User Interaction solution in UbiCollab, is the glue between user actions and the underlying component model. Acting as a User Interface manager, it supplies to other modules a plugin mechanism to let them publish their proprietary UIs engines as soon as they are discovered by the RD subsystem and without charging any configuration process to the user. Moreover this approach let UI can be developed without extensive coding since new UIs mechanism can wrapped in the existing modules.

In fact an UC component can expose more than one user interface[2], for examples it may have a simple GUI adapted for small phones display likewise an enhanced GUI with voice cognition interface for more capable devices (figure 3.5).
The way in which the UB eWorkbench presents and internally handle the UIs embedded in the bundles make use of two abstract concepts: views and perspectives.

### Views

A view is the component of the bundle directly involved in the user interac-

[2]Requirement AR-UI-02

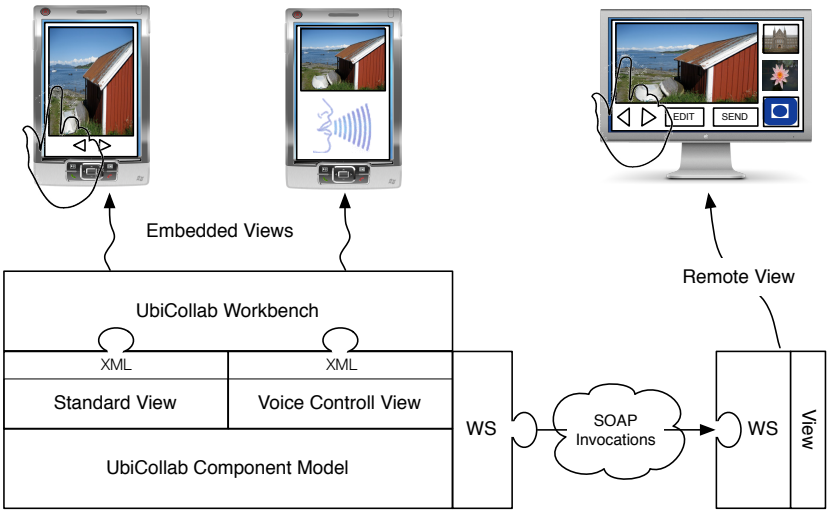



**Figure 3.5:** An UC component which exposes multiple UIs

tion. Each view is composed by a GUI and an interaction mechanism with it that could be, for instance, touch based, voice based, gesture based, or another proprietary technique.

All the views stand for child view of the eWorkbench, they are embedded in the component package and connected to the eWorkbench by an XML file (figure 3.6), the eWorkbench decides which view has to be activated and shown to the user due to the device hardware features and user preferences.

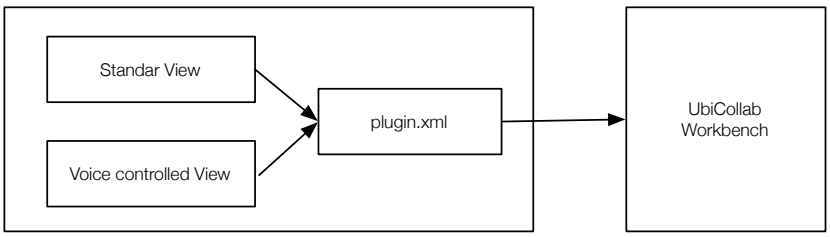



**Figure 3.6:** Views - eWorkbench connection

### Perspectives

A perspective is a set of views that belongs to the same scope, for examples all the UIs provided by the Resources Discovery Plugins belongs to the Resources Discovery Perspective, in the same way of all the UbiCollab Applications

belong to the application Perspective.
As we wrote in the problem definition, to do an action like discover a new resource we provide different user interactions mechanisms, called *Discovery Gesture*, these hence are implemented as views in the resource discovery perspective of the eWorkbench, as outlined in figure 3.7.

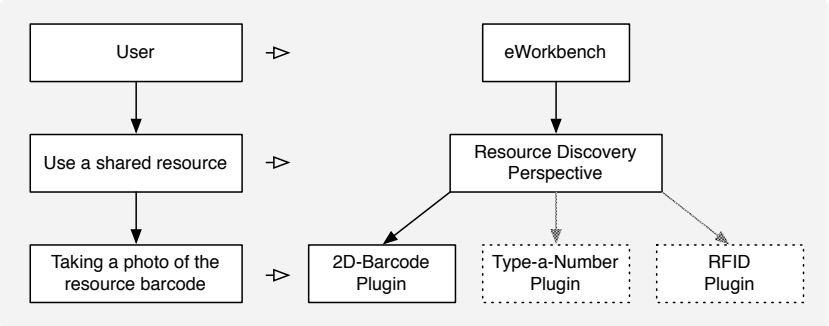



**Figure 3.7:** User behavior - Perspective matching

Formally perspectives are conglomerations of views that come from one or more components which map predefined user tasks with the system (figure 3.8).

We defined three perspective:

- Resource Discovery Perspective
- Service Domain Manager Perspective
- Applications perspective

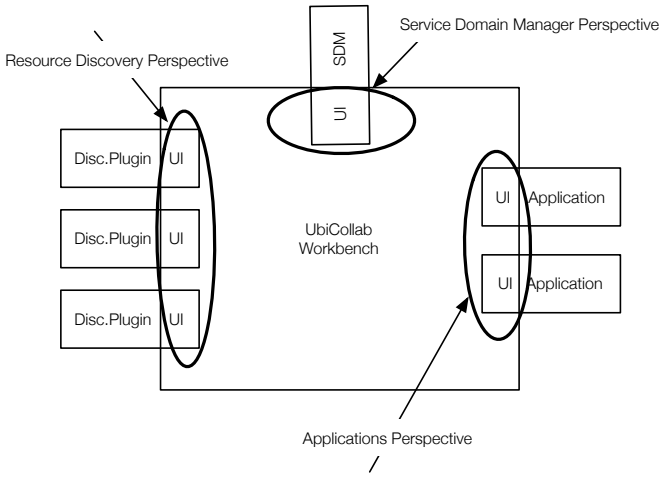



**Figure 3.8:** Perspectives

## 3.3 Integration with external frameworks

### 3.3.1 Integration with ASTRA

ASTRA[3] (Awareness Services and System - Towards theory and ReAlization) is an european founded project (IST-FP5/6). The aim of ASTRA is to define a framework for supporting the conception and the design of Pervasive Awareness systems, specifically those that are intended to support social relationship. The envisaged framework consists of theories and technological solutions, in order to develop a theory to guide the design of systems for supporting social communication. In addition the project develops a platform based on a Service Oriented Architecture, tools and applications that can support communities to create, adapt and appropriate their own awareness system.

ASTRA is lacking of a proprietary service discovery manager, thus we decided to provide to ASTRA UbiCollab's resource discovery capabilities. As result of that ASTRA applications can use resources connected to UbiCollab to get information about the physical environment and drive them to generate

[3]http://www.astra-project.net/

notifications to the user.

The integration between UbiCollab and ASTRA aims to improve ASTRA's capability in resource discovery fields providing a user friendly way to add external devices to an ASTRA node and allow access to the resources already installed in the UbiNode. According with the SOA, UbiCollab proxies won't be deployed in the ASTRA node but rather UbiCollab will provide to ASTRA information about the proxies installed in the UbiNode and a WebService interface capable to drive them.
The core engine of the UC side of the integration has been implemented in the *AstraConnector*, an module that guarantee the communication between the two platform.

**It acts on two side:**

1. **Exporting a representation of the proxies already present in the UbiNode on a system startup and providing real-time notifications about the proxy will be discovered next by a *Discovery Gesture*.**
2. **Listening for WebService invocations from an ASTRA node containing actions that will be delivered to a proxy and thus will be performed by an external resource.**

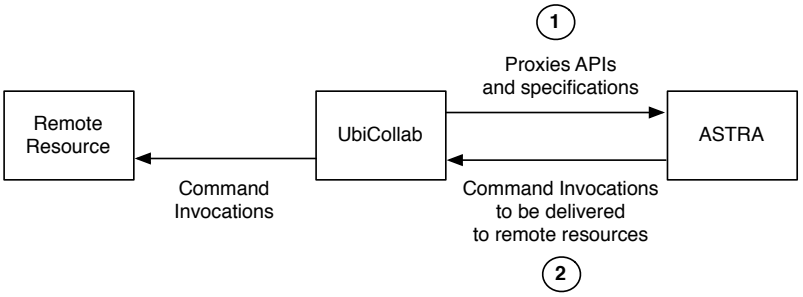



**Figure 3.9:** High Level Design of the UbiCollab-ASTRA integration

The implementation of the *AstraConnector* module will be analyzed in chapter 4.

## 3.4 GUI Design Guidelines

UbiCollab aspire to be platform independent and provide multiple interaction approaches, but for our proof-of-concepts applications development we had to choose a family of the mobile devices for the deployment and a User Interaction mechanism. First off we chose to focus on touch-based Graphical User Interfaces as interaction mechanism and consequently we chose smartphones (which usually comes up with touchscreens) as deployment platform.

### 3.4.1 Target Platform

The target devices which currently host the UbiCollab platform, in the UbiNode shape, are smartphones running Windows Mobile Professional 6.1. According with Microsoft [11] WMP6.1 allows screen resolution up to 480x800 pixels (WXGA), thus I focalized the design on this screen resolution that would probably became a standard for smartphones and PDAs in the close future.

The screen size (or better the screen coverage area) combined with screen resolution is a very important aspect, not just for designing the more usable icons and buttons sizes but even because it determines what kinds of gestures (use fingers instead of the PDA stylus, one or two hand, etc) are appropriate or even possible to have on the target device.

The device chosen to host our platform is the HTC Touch HD, which has WXGA screen resolution and 3.8" screen size, see the Appendix C for the full technical specifications.

### 3.4.2 Finger-Operated vs Stylus-Operated Approach

Starting to design GUIs has been evaluated which kind of user gestures would have been available for the target device, and first off we had to decide be-

tween adopting a finger-based interaction with the device or the stylus-based interaction commonly used in windows mobile platforms. I established my evaluation drawing on the "SAP Interaction Design Guide for Touchscreen Applications" [12] which reports comparison criteria between finger and stylus input as summarized in figure 3.10.

| | Finger-Operated | Stylus-Operated |
|---|---|---|
| **Interaction** | Tap, drag | Click, double-click,drag |
| **Operations** | Point, select | Point, select, define path (start and goal, path): drawings, gestures, handwriting |
| **Speed** | High | High |
| **Accuracy** | Low | High (comparable to mouse) |
| **Size of Controls** | Large | Small (as with mouse) |
| **Text Entry** | Not recommended | Through handwriting |
| **Number Entry** | Through selection | Through handwriting or selection |
| **Initiation of actions** | Through point-and-tap (pushbuttons) | Through point-and-clic (pushbuttons), through selection (e.g. dropdown lists), through gestures |
| **Preferred Interface** | Point and tap interface | "Standard" GUI possible or optimize interface for pen, point-and-click interface if speed is required |
| **Environment** | High speed, low accuracy, in "aggressive" environment pen is disturbing (taking up the pen, dirt, possibility of loss) | High speed (not mandatory), high accuracy, pen usage possible |
| **GUI Elements** | Pushbuttons, controls for display selection of data, graphics | Nearly all standard GUI elements may be used (but not all are optimal), avoid scrollbars, pulldown-menus (at least these should be spring-loaded) |

**Figure 3.10:** Finger-Operated vs Stylus-Operated Approach (data from SAP research)

Considering our Discovery Gesture paradigm and our target scenarios (see Appendix A) we can assume that what we essentially need is a GUI where

the point-and-tap[4] on few screen buttons would be the most used gesture. We don't need to do heavy data entry operations with the device and the number entry operation required by the Type-a-Number discovery gesture can be handled drawing the needed digits as screen buttons instead of using a full keyboard that could be bewildering. Looking to the fact that UbiCollab has to support a wide range of population, including elderly people; we think that the chance to avoid the use of a small stylus that could be lost and may be difficult to manipulate for someone has definitely to be taken. Furthermore the finger-operated approach also allow the use of just one hand, even wearing gloves[5]. For the illustrated reasons we chose to develop fingers-operated GUIs.

### 3.4.3 Design Patterns

The minimum size of buttons and other interface elements is determined by the size of an adult finger. According with MIT's researches [13] adult fingers typically have a diameter of 16 mm to 20 mm, children's and teens' fingers may be smaller; elderly, disable and obese people may have misshapen or larger fingers. When interacting with a touchscreen usually the pad of the finger is used instead of the tip. Fingertips are narrow, only 8-10mm wide. Because of this small surface area, humans usually push buttons at an acute angle using the pad of the finger, not straight on using the tip of the finger [7]. Finger pads are wider than fingertips, typically 10-14mm (figure 3.11).

Gloves can make it difficult to use GUIs, so in climates that often necessitate to wear gloves, as the norwegian one, we should keep on mind that our GUIs

[4]The point-and-tap paradigm substitutes the point-and-click paradigm for hand gestures, the finger's tap is considered the new mouse click

[5]This is not valid for touchscreens that use capacitive sensor panels technology, whose display are coated with a material that stores electrical charge. When a user touches the screen a portion of the charge is transferred to the user, decreasing the panel's capacitive layer and thus triggering a touch event. For this reason this principle doesn't work if hands are electrically isolated by gloves.

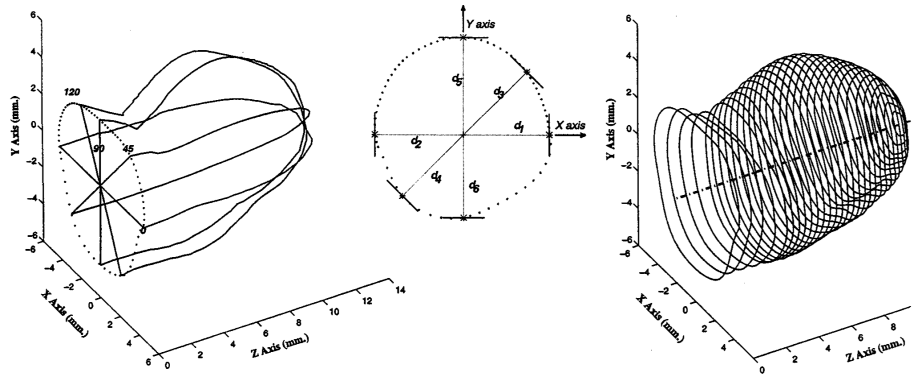



**Figure 3.11:** The algorithms for 3D reconstruction of a fingertip (Courtesy of MIT)

probably will be handled with gloves on for a considerable amount of time.

### Buttons and Targets

As our GUIs will be mainly used for point-and-tap operations, most of the screen area will be filled with pushbuttons; therefore buttons play an important factor in the design.

The range for what counts as an acceptable target size varies widely, but we considered as one reasonable guideline that the target should be no smaller than the smallest average fingertip which, as said before, is rounded up to 1cm in diameter or 1cmx1cm in square.

However, what 1cm sized target is translated into the pixel domain depends on the pixel density or pixel per inch (PPI). Pixel density is a measurement of computer display resolution, PPI is related to the size of the screen measured in inches and the number of pixels available (screen resolution). You can compute the PPI by dividing the width (or height) of the display area in pixel by the width (or height) of the display area in inches. The higher is the PPI, the larger your interface elements will have to be to create suitable touch targets. To calculate the ideal size for a button we used the equation 3.1 [7]:

$$target = target\_size\_in\_inches \frac{screen\_width\_in\_pixels}{screen\_width\_in\_inches} \tag{3.1}$$

### Screen Layout

Designing a layout for elements rendered on a touchscreen is quite different from designing a normal GUI for a mouse operated desktop computer, due to the following differences:

- Our finger pads, unlike a mouse cursor, don't float transparently in space; the rest of the finger, the hand and the arm will likely cover up some part of the interface while the user is touching it, especially the part of the screen immediately below what the user is interacting with. For this reason placing menus and controls at the bottom of the screen instead of in their traditional place at the top is helpful to prevent screen coverage and involuntary button clicks (figure 3.12).

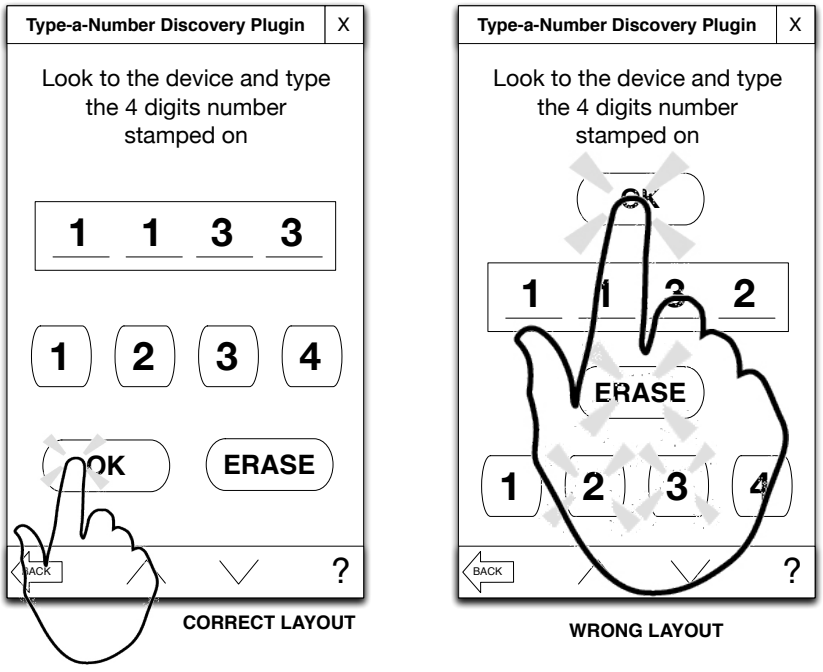



**Figure 3.12:** Layouts comparison

- With traditional input devices such as mouse or trackball, it makes

good sense to place targets such as menu items on the edges of the screen so that the hit target becomes huge because the user cannot overshoot it as the cursor stops at the edge of the screen. With touch interfaces users usually don't drag their finger across the screen as they do with a cursor but they will likely lift their fingers and place them from target to target, therefore put targets in screen corners doesn't improve usability.

- Touchscreens get finger oil (and dirt) as well as fingerprints and smudges. Dark backgrounds and color patterns (popular on many default smartphone interfaces) makes fingerprints even more visible and disturbing, the use of brights background colors helps to mask this issues.

- Unlike a mouse does, a touch event does not return a mechanical feedback to the user's fingers, therefore implementing a notification of the touch event by a sound alarm or vibration can be really useful to have a physical feedback and thus to avoid triggering not intentional operations.

- As well of common interfaces we have to pay attention to the mutual positions of buttons drawn on the screen; the Fitt's law (Equation 3.2) help to define it. Fitt's law simply states that the time it takes for a user to reach a target by pointing it (with a finger or a mouse) is proportional to the distance to the object divided by the size of the object. Thus a larger target that is close to the user is easier to point to than a smaller one farther away; for this reason buttons and other visual targets need to be designed in a way that important and preferred operations are reached by gestures which minimize the distance for reaching them by fingers.

Fitt's Law[6]:

$$MT = a + b \log_2 \left( \frac{2A}{W} + c \right) \tag{3.2}$$

## 3.5 Platform Abstraction Layer

### 3.5.1 Components Standardization

An UbiCollab module can span between a wide range of tasks. We grouped them into task domains with an assigned fixed identifier, each component has to belong to one of these domains. However, since all the module share similar features as well as mandatory classes used to wire them with the frameworks, in Chapter 4 (Implementation) we will also provide a draft of the internal structure highlighting mandatory and optional units. We hope that this kind of standardization, combined with the standardized buttons and frames provided by the UIToolkit, will help developers to pursue best practices developing their modules. Each component class own a name and a short name which will be used to technically identify components by developers. Classes names and name abbreviations are reported in figure 3.13.

**Component classes:**

- **Core Component:** is intended to be a mandatory modules which shapes the basic functionality and provides support to other components; therefore has to be included in every UC distribution. Instances

[6]* MT is the average time taken to acquire the target.

* a and b are empirical constants determined through linear regression.

* A is the distance from the starting point to the center of the target.

* W is the width of the target measured along the axis of motion (how close to the target you need to get to count as acquiring it).

* c is a constant which is either 0, .5, or 1, depending on the specific environment.

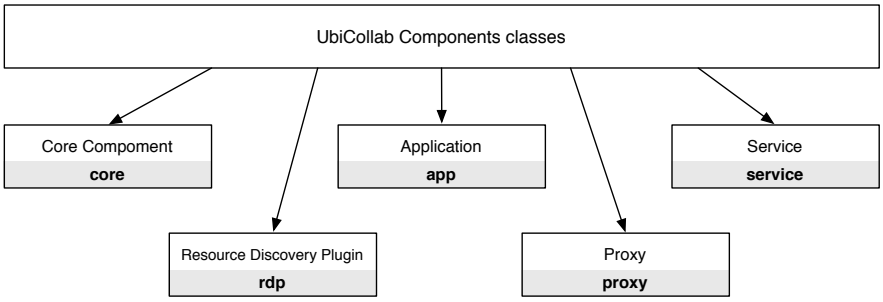



**Figure 3.13:** UbiCollab Components Classes

of core components are the eWorkbench and the Resources Discovery Manager

- **Resource Discovery Plugin:** is intended to be a plugin for the Resource Discovery core component which implements a specific discovery gestures. They fundamentally do the same task: feed the Resource Discovery with the location of the proxy service, but they operate in different ways like typing a number or reading a RFID tag.

- **Application:** an application is a piece of software directly operated by the end-user, like an Instant Messaging application or text editor. It can be developed by third-party company, independent developers or students (thanks to the UbiCollab open source license). Complex applications can use a proxy to get informations and drive a remote resource. The Image Viewer app developed as contribution to this thesis takes steps in this way; other examples could be an application for controlling a coffee machine via a X10Proxy[7] or an application which monitors blood pressure through a sensor embedded in a shirt.

- **Proxy:** As detailed explained in the introduction, a proxy is a sort of driver that let internal application talk with remote embedded device. It need at least one RD Plugin in order to be discovered and usually doesn't provide an user interface since is directly controlled by an UC

[7]An X10Proxy is a proxy service for the X10 protocol which allow devices remote control using the common power-grid.

application which furnish it. As contribute to this thesis have been developed proxies for shared screen resources.

- **Service:** A service is a third-party software that controls a remote device. It can be a bundle inside the UC framework, as are the services developed for out tests, but is more intended to be a proprietary software developed in any language and embedded in the device by the manufacturer. We don't care about service implementation since it has to adhere and fulfill the WebService standard exposing a wsdl[8] document which is the only interface needed for building a proxy for UbiCollab. For this reason a service can be any software from a component in another UC distribution to a internet weather forecasts service or an Amazon book search Webservice.

## 3.6 Platform Summary

Platform components and available tasks are schematically presented in figure 3.14.

The reported Platform design show a basic UbiCollab distribution which consists in the platform components for Resource Discovery and Service Administration, an Application that make use of remote files and shared resources, a proxy services repository.
A generic interaction with the system consists in discovering a shared device and starting to interact with it via the proxy and through an application.
This operation goes towards the following steps:

1. A resource is advertised with three different tags: an RFID, a QR Barcode and a Label with a 4-digits code. The user choose which wants to use for the discovering operation activating a RDPlugin. The RDPlugin retrieves the URL of the service and pass it to the RDM.

[8]Web Services Description Language: http://www.w3.org/TR/wsdl

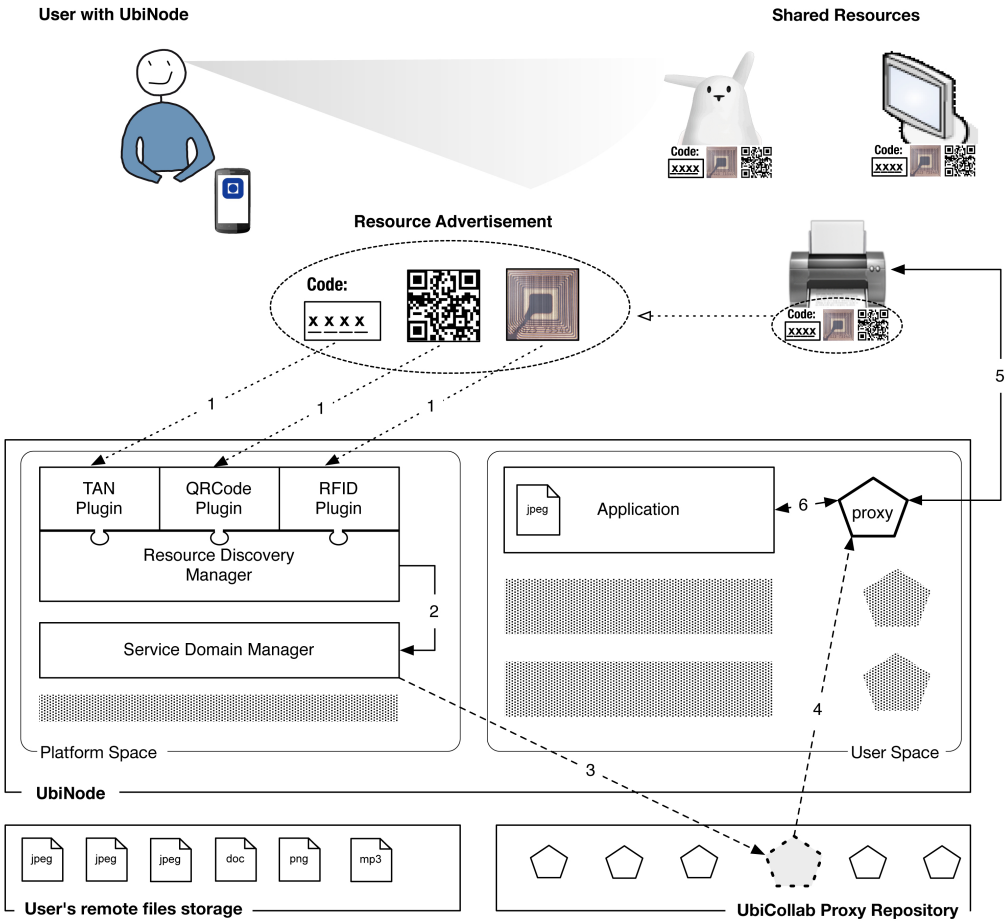



**Figure 3.14:** Platform Summary

2. The RDM pass the URL to the SDM throughout a SOAP invocation.
3. The SDM connect to the proxy repository, then it fetches and downloads the proper proxy.
4. The Proxy is installed and activated.
5. The proxy takes care to establish a connection with the remote resource which is thus now available to the applications.
6. The Application starts to use the shared resource via the proxy

A scenario which revises these concepts will be outlined in the next section.

## 3.7 Scenario

We elaborated a test scenario for the described architecture

**The scenario is the following:**

*Bjrn is invited to held a presentation in a foreign university Meeting Room equipped with shared screens. The available shared screens are advertised by a four-digits code and a 2D Barcode.*
*Bjrn has never been in that room and doesn't know the specification of those displays, but he would to use them to show slides and pictures to the audience.*
*At home he uploaded slides and photos on his web personal space and he left home just with his UbiNode smartphone.*

*On the presentation day he reviews the speech in his hotel room looking to the slides showed on the smartphone with UbiCollab Image Viewer application; then he reaches the presentation room and finds two UC shared screen of different sizes. One is a tablet PC and one is a huge LED screen.*

*He decides to use them to share the slides with his audience using the smaller screen to show pictures and the bigger one to show a text related with the content.*

*He takes out from the pocket his UbiNode and runs the UbiCollab platforms interacting with the system via a touch interface. He finds on both screens an advertising label reporting the two Discovering Gestures available for those devices: a numeric four-digits code and a 2D Barcode.*
*He chooses to discover both screens by typing a code because his smartphone doesn't have a camera.*
*The UC Resource Discovery Manager installs the UC proxies for those devices, establishes a communication with them and notify the user about the outcome of the operation.*

*Finally Bjrn runs the UC Image Viewer application he used earlier to review the slides, now the application recognizes and notifies the user that two new shared screen services are now available and start to communicate with them. The two shared screens get activated and start to display the contents of the presentations in a proper way. Bjrn controls and browse the slide to display interacting with his smartphone and the contents showed on the shared screens are automatically updated.*

This scenario will be evaluated in Chapter 5.

# Chapter 4

# Implementation

In this chapter we describe how the solution proposed in Chapter 3 has been implemented, components development will be described in details. Applications, Proxies and WebServices have been developed as proof-of-concepts of the solution proposal and there will be evaluated in the next chapter. Before starting with the in-depth analysis a short introduction is given and some important aspects are illustrated.

Because UbiCollab components have to be published on Sourceforge and all the platform have to be released as Open Source software, we try to make use, even for the third-party components involved in the project, of open source tools. At the present time just the Java Virtual Machine we run is under a commercial license but, as explained later, we have explored several possible alternatives.

In section 4.1 the UbiCollab implementation is defined and each stack tier is explained in details, possible tier implementations are given as well as justification of the choices made.

In section 4.2 the inner component architecture based on functional units is described tier-by-tier

In section 4.3 we will present an overview of all the implemented components,

grouped by category, and where they have been deployed for testing. Finally we will describe each own component implementation.

We will draw UML diagrams of the main functionalities; because the large number of class-files distributed in several components, complete class diagrams have not been created for all of these. The focus is to highlight the architecture keypoints, therefore diagrams made includes just objects needed to understand its own logical functioning. All the implemented components code has been documented using JavaDoc and sources are freely available on UbiCollab SourceForge website[1]. All the components have been fully developed by the writer except for the eWorkbench that has been built on a draft component jointly developed by the Eclipse foundation and Nokia.

## 4.1 The UbiCollab Implementation Stack

The UbiCollab Platform runs on the top of a stack involving different technologies (figure 4.1).

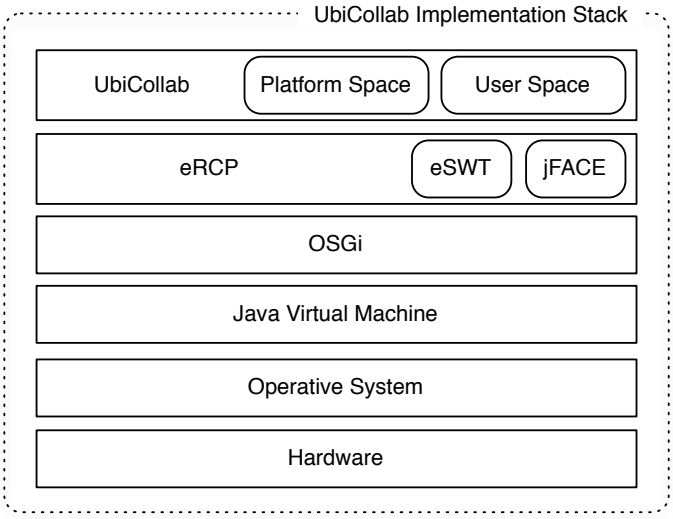



**Figure 4.1:** The UbiCollab Implementation Stack

[1]http://ubicollab.svn.sourceforge.net/viewvc/ubicollab/

This stack fulfill requirements in mobility area. A complete evaluation of other requirements will be presented in Chapter 5 (Evaluation).

Subsequently the stack is analyzed tier by tier.

### 4.1.1 Hardware

Ubicollab is implemented pursuing a scalar paradigm (figure 4.2) which takes and shares advantages with the underlying Service-Oriented Architecture approach.

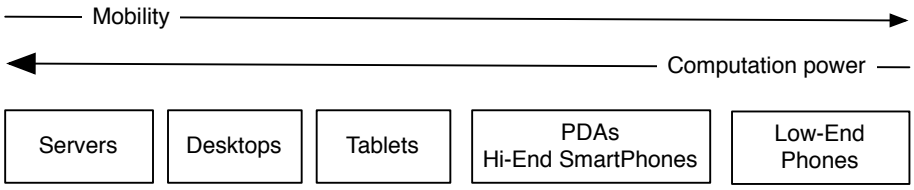



**Figure 4.2:** Device Supported

Thanks to SOA we can deploy each platform module on devices that fit hardware constrains and, at the same time, user requirement in mobility and usability areas.

### 4.1.2 Operative System

Since UbiCollab aims to be platform independent we would like to say that it runs over all the operative system for which an implementation of the Java Virtual Machine exists. This is quite true regarding the core engines but it's not concerning user interaction interfaces. Actually, considering GUI usability issues there's an insider trade-off between portability and usability: we can design a GUI with a proprietary look and feel[2], OS independent, or we

[2]In GUI design, look and feel is used to depict aspects of its design, including elements such as colors, shapes, layout, and typefaces (the "look"), as well as the behavior of dynamic elements such as buttons, boxes, and menus (the "feel")

can use the OS native l. and f. to render our GUI components.
We adopted the second approach thinking that our platform should be integrated as much as possible with device operative systems, in view of the fact that users are used to interact with it and would find less painful to learn how to use UbiCollab if, for example, popup notifications and screen buttons are provided in a way they are used to recognize and interact with. This approach has drawbacks in portability domains since a java implementation of system widgets[3] is needed. As we will see later, the use of eRCP/eSWT UI framework imply that, in order to provide native widget to third party applications running over the OS -as UbiCollab is- a "bridge" or to be more precise an implementation of system widget for the UI framework, needs to be provided between the OS and the UI framework (figure 4.3).

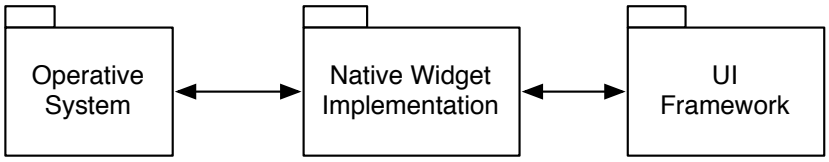



**Figure 4.3:** Native Widget Implementation Dependencies

Using Eclipse eRCP/eSWT runtimes (for the reasons explained latter) we need to run Windows, for desktops, or Windows Mobile and Symbian OS, for mobiles; given that at the present time implementations of native widget are provided just for those Operative Systems.

### 4.1.3 Java Virtual Machine

The java programming language grow up a lot in the last years becoming one of the most popular development environment. It moved from supporting

[3] A widget is an element of a graphical user interface that displays an information arrangement changeable by the user, such as a window, a text box or a button. The defining characteristic of a widget is to provide a single interaction point for the direct manipulation of a given kind of data.

consumer electronic devices, purpose for whose was created, to support a wide range of platforms: from servers to mobile devices, passing through desktop PCs and ending with smartcards.
There are mainly four technology editions of the Java Platform, according with a scalar approach, as presented in figure 4.4 :

- **Java Platform, Standard edition (Java SE):** which is designed for desktop applications
- **Java Platform, Enterprise edition (Java EE):** a superset of Java SE that support scalable, transaction-oriented, and database-centered enterprise programming.
- **Java Platform, Micro edition (Java ME):** specification of a limited set of runtimes and APIs for embedded consumer devices, such as mobile phones, PDAs and other devices that are constrained from supporting a full Java SE or Java EE implementation
- **Java Card:** a small Java framework including security and remote invocation APIs intended to develop applications deployed on smartcards. It is widely used in SIM cards (used in GSM mobile phones) and ATM cards.

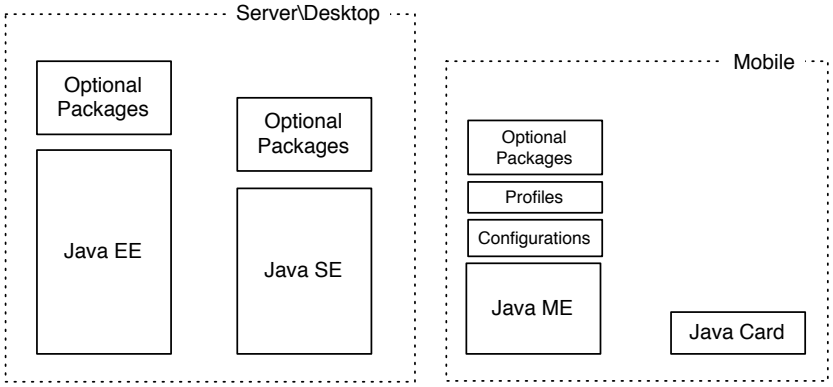



**Figure 4.4:** Java Distributions

An application developed in compliance with the ubiquitous computing paradigm should take advantage from all four dictions in different scenarios. Our focus

will be on Java ME because our platform is mainly deployed on mobile devices and all the applications designed for Java ME will run at the same way for the SE and EE editions, thanks to the backward compatibility of Java distributions[4].

Unlike JSE and JEE, JME is not a piece of software, nor is it a single specification. Because JavaME spans such a variety of devices, it wouldn't make sense to try to create a one-size-fits-all solution, therefore Java ME is divided into *configurations, profiles* and *optional packages*.
Devices implement a complete software stack which usually consists of a configuration, a profile, and optional packages (figure 4.5).

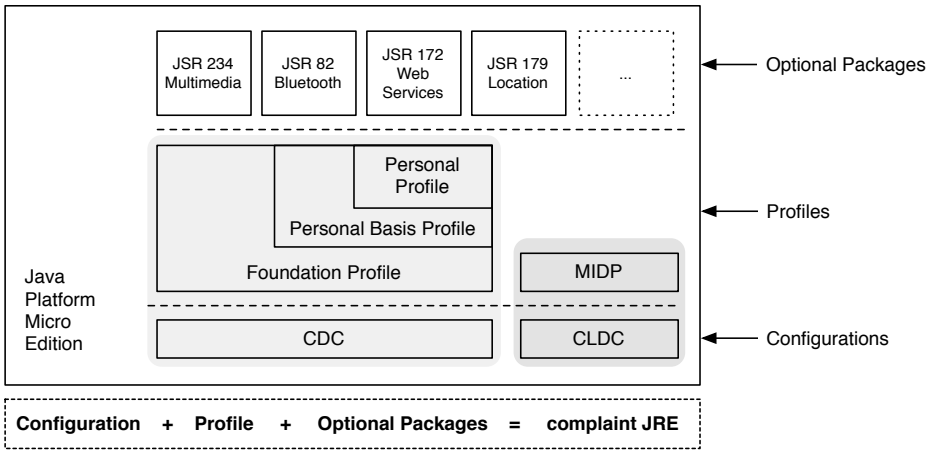



**Figure 4.5:** Java for Mobile

A *configuration* defines the core functionality of the platform runtime environment; this includes the Virtual Machine and a set of core classes derived from the Java SE platform.
At the heart of Java ME are two configurations, targeting different family of devices:

[4]This is mostly right except for some optional components addressing the same specification but implemented in different way for mobile and desktop platform, for instance the Mobile Media Extensions (JSR135) for mobiles and the Java Media Framework (JMF) for desktops. Moreover to address the stricter limitations of devices, Java ME sometimes replaces Java SE APIs and adds new interfaces.

- **CLDC** (Connected Limited Device Configuration): supports smallest devices such as cell phones, two-way pagers and low-end PDAs. Technically speaking, devices with 16-bit or 32-bit processors, at least 160KB of persistent memory and at least 32KB of volatile memory.
- **CDC** (Connected Device Configuration): supports more powerful connected devices, such as high-end PDA and Smartphones as well as sophisticated embedded devices. Technically speaking, devices with 32-bit processor, at least 2MB of volatile memory, 2.5MB of persistent memory and network connectivity

CDC includes all the classes defined by CLDC, including any new ones not included in the Java SE platform, since they are designed for addressing mobile constrains in communication area.

Right above configurations there are *profiles*. A profile, as well as optional packages, builds on a configuration providing classes for managing applications life-cycle, driving UIs, accessing data locally and over the network. Is a way to include in the distribution a set of standardized optional classes addressing domain-specific functionality that most or all devices in a class need. At this writings, there are four profiles, one based on CLDC and three on CDC, as schematically presented in figure 4.5:

On the top of the stack are optional packages. These can be seen as profile extension since they provide support in relatively narrow areas of functionality that some devices and applications need but other's don't, such as messaging, multimedia and location service. One example of an optional package is the Multimedia Support (JSR234), which provides access and control to smartphone multimedia resources, like phone cameras. This optional package could be implemented alongside virtually any combination of configurations and profiles. All JavaME optional packages are defined by the JCP[5], making

[5]Java Community Process, a open community-based standards organization (under Sun Microsystems authority) with a formal process for defining and revising Java technology specificatio

them standard APIs

As it will be proved afterwards UbiCollab needs, in order to run, a Foundation Profile, CDC Virtual Machine enhanced with some optional components, as listed in figure 4.6

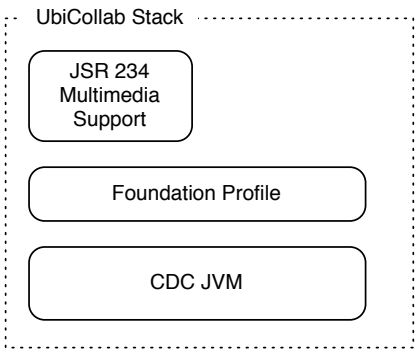



**Figure 4.6:** Java Stack used in UbiCollab

The use of different configurations has been investigated but finally this one resulted the only full compatible with our platform. In truth all the configurations below CDC doesn't provide enough resources to run the OSGi framework; on the other hand all the profiles higher than Foundation already include a support for UI technologies like AWT for the Personal Profile or even SWT for some stack's implementation[6], these components crash against our UI framework implementation, making the whole stack unstable.

### 4.1.4 JVM Implementations

Nowadays a lot of Virtual Machine implementation, both open source and commercial, based on Sun's specifications are available for the deployment. Seeking the best solution for our platform in terms of performances and compatibility, we filled the comparison chart reported in figures 4.7 and 4.8. Important features as well as weakpoints are highlighted.

[6] As for the IBM implementation of the CDC stack

| Java | IBM J9 | Esmertec Jbed | Mysaifu JVM | Sun phoneME |
|---|---|---|---|---|
| **Supported OS** | WinCE 2.11, WM2003/6/6.1, Linux | WM5/6 | WM2003/5/6 | WM 2003/5/6, Linux, OpenWRT |
| **JVM Compatibility** | CDC 1.1/ Foundation profile Personal Profile | CDC/Personal Profile | JavaSE 5 | CLDC/MIDP CDC/Foundation Profile, Personal Profile |
| **Additional supported package** | Yes: JSR 75,135 | Yes: JDBC and JRMI packages | No | Yes: JSR 75, 120, 82, 135, 184, 205, 226 |
| **UI implementations included** | AWT, SWT | AWT | AWT | AWT |
| **Supported Hardware Architectures** | ARM arch. (WM & Linux), x86 arch. (Linux), PowerPC | ARM arch. | ARM arch. | ARM arch. |
| **KeyPoints** | High Reliability Technical Support from IBM Eclipse recommended JVM for eRCP | Proprietary acceleration technology. Really small footprint (~4mb) | JavaSE 5 full compatibility Small footprint (~10mb) | Full JavaME specification support Open Source license |
| **WeakPoints** | Windows Mobile version is commercial | Commercial. Foundation Profile not available | Unstable, crashed when tested with UbiCollab | Relatively new and thus less documented and tested |

**Figure 4.7:** JVM Implementation comparison chart

| Java | Jalimo (Cacao JVM) | NSI CrEme | Sprint Titan |
|---|---|---|---|
| **Supported OS** | Linux (Maemo and OpenMoko distribution) | WM6/6.1, Linux | WM6 and later |
| **JVM Compatibility** | Java SE 5 | CDC1.0/Personal Profile | CLCD/MIDP CDC1.1/Foundation Profile |
| **Additional supported package** | No | No | JSR 118, 135, 75, 179, 120, 232 |
| **UI implementations included** | eSWT (unofficial) | AWT, Swing support | eSWT |
| **Supported Hardware Architectures** | i386, x86_64, Alpha, ARM, MIPS641, PowerPC 32/64, S390, SPARC64. | ARM arch. | ARM arch. |
| **Architectures** | Full Java SE support Released under GPL license Wide support from opensource community | Tested with Eclipse SWT Full support to JNI | Designed for eRCP Complying with JSR232 (Mobile Operational Management-OSGi) standard. Proprietary development tools |
| **WeakPoints** | Just few mobile devices support linux distributions | Commercial | Only licensed for Sprint branded smartphones |

**Figure 4.8:** JVM Implementation comparison chart (2)

In accordance with considerations expressed in the comparison chart, we chose to focus on the two Virtual Machines which express best reliability and compatibility with other third-party technologies employed in UbiCollab.
At the present time IBM J9 is the most reliable and tested JVM available and it works well with UbiCollab but IBM is discontinuing the free version moving to the integration of it in its commercial product; that is not good step for a project UbiCollab since it aspires to work in a opensource environment.
Sun's phoneME project, despite it suffers from some youthness problems and is not widely tested enough, has a really bright roadmap [14] and, as it will be proved in Chapter 5, it has expressed good performances.
Sprint Titan is also a really promising solution for our platform. It is a complete mobile framework that already include OSGi and eRCP tiers and is fully complain to the Mobile Operational Management (JSR232) that most likely will become a standard, for mobile modular software distributions, in the near future. If the Titan project will earn enough notoriety to cross US borders and being also licensed for the rest of the world it will be certainly a perfect running environment for UbiCollab.
Benchmarks of the two quoted JVM have been made and reported in Chapter 5 (Evaluation).

### 4.1.5 OSGi

Modularization and Services are two cornerstones for the UbiCollab platform. In order to implement these concepts in our software components the plain Java programming language is not efficient enough.
The source of concerns coding with traditional Java is that the global, flat, classpath and the indeed absence of dependency management do not properly fit the requirement of a Service Oriented Architecture.
OSGi provides a solution to these issues since is both [15] a programming model to develop Java applications from modular units *(bundles)*, decoupled through *service interfaces*, and wired in a runtime infrastructure for control-

ling bundles life cycle. OSGi improves modularization deploying each bundle in a JAR file with an enhanced manifest used to wire it in the framework. All that allows developers to dynamically manipulate bundles: new bundles can be added, existing bundles updated or removed all at runtime, without rebooting the Java Virtual Machine. This means, in UbiCollab terminology, that is possible to download, install and start to use a proxy service without rebooting the UbiNode, according with AR-UI-07[7] requirement. OSGi maintains consistency across modules by keeping track of the dependencies between them, at the same time it makes them loosed coupled by arranging for each module an its own classpath, separated from the classpath of all others module. This method imply that the framework provides a separate class loader for each bundle, therefore just classes and resources inside the JAR file are loaded. This is the core secret that stand at the ground of OSGi: in standard Java class loaders are arranged in a hierarchical tree, loading requests are a delegated upwards, and classes cannot be shared horizontally: OSGi moves to a network-like paradigm where dependencies between modules can bee seen more like a provider-user relation instead of a parent-child one, and loading request are delegate from one bundle's class loader to another's based on the dependency relationship between the bundles (figure 4.9).

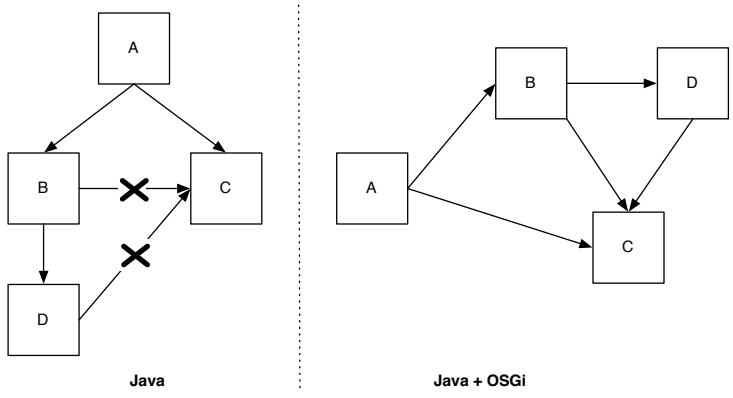



**Figure 4.9:** Java vs. Java + OSGi Dependency Management

[7]AR-UI-07: System's components just installed must show their UI mechanism without rebooting the platform or the device.

We can easily realize that this kind of approach really shapes our idea of a centralized service-oriented architecture with loosely coupled services, since in the OSGi model any Java class can be published as a service to be used by other bundles (services) in the system.

OSGi is a standard defined by an Alliance of around forty companies, including IBM, Motorola, Oracle. The current standard version, release 4.1 (R4.1) has been shipped in March 2007 and especially improves the wide used release 3 (R3) shipped in 2003.
OSGi specification are freely available and several independently implementations both commercial and opensource are available. Our research interest is centered on implementation that obey open licenses, the most popular are:

- **Equinox: [16]** is the widest deployed OSGi framework and is the core runtime for most of the Eclipse foundation products. It's born to work together and support eRCP UI framework and this combination can be found in many custom application, desktop and mobile, as for instance the IBM WebSphere suite[8]. Equinox implements Release 4.1 of the OSGi specifications and is licensed under the Eclipse Public License (EPL) [19]

- **Knopflerfish: [17]** is a popular and mature implementation of both OSGi Release 3 and Release 4.1 specifications. It is developed by Makewave AB and licensed under a BSD-style license. There is even a commercial version of this distribution called Knopflerfish Pro.

- **Felix: [18]** is an implementation of the OSGi release 4.x by the Apache group. It is designed particularly for compactness and ease of embedding and it feature one of the smallest footprint for Release 4 implementations. It is licensed under the Apache License Version 2.0

- **Concierge: [19]** is a very compact and highly optimized implementation of OSGi Release 3. This makes it particularly suited to resources-

[8]IBM WebSphere Software - http://www.ibm.com/software/websphere/

constrained platforms such as mobile phones. Concierge is licensed under a BSD-style license.

The version of UbiCollab which I started to work with was running over the Knopflerfish implementation, during the GUIs implementations we decided on move the platform to Equinox, since it's the most natural running environment for the eRCP framework we adopt to implement our user interfaces.

### 4.1.6 eRCP/eSWT

Eclipse *Rich Client Platform (RCP)* is a platform for building and deploying rich client applications [20], it let multiple applications run in a single JVM using OSGi and allows developers implement native GUI applications to a variety of desktop operating system such as Windows, Linux and Mac OS. The *embedded Rich Client Platform (eRCP)* aims to extends RCP features to mobile and embedded devices. Generally eRCP APIs are quite similar to RCP ones, but more lightweight. eRCP APIs are created using a subset of RCPs further modified to fit constrains and features of the embedded devices. Modification mainly concerns adapting it to run over constrained resources which embedded devices have, like: small screen, reduced amount of memory, small keyboard, etc...
Because eRCP is basically a subset of RCP, applications developed to run on embedded devices automatically run on the desktop platform. As eRCP applications will likely be optimized for small screens, display on a desktop might not be optimal, but the application should be functional; anyway by the high number of shared components is really fast to upgrade a mobile application to fill the desktop model. eRCP, in order to run, requires at least CDC Foundation Profile Java Virtual Machine and an OSGi implementation; the Eclipse foundation recommends equinox for OSGi, and the IBM J9 as JVM.

The eRCP is made up of the following components (figure 4.10):

- embedded Standard Widget Toolkit (eSWT), with Core, Expanded and Mobile extensions
- eJFace
- eUpdate
- eWorkbench

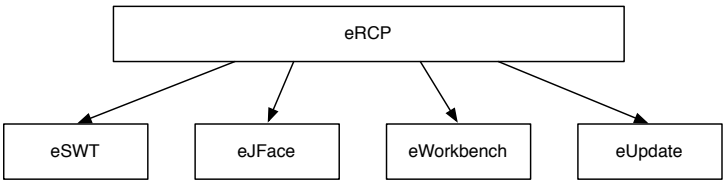



**Figure 4.10:** Eclipse eRCP

### eSWT

The embedded Standard Widget Toolkit-eSWT (a subset of the well-known Java graphic Standard Widget Toolkit-SWT) is a technology that can be used to develop native-looking applications for a variety of mobile phones. It comes as part of eRCP framework but is designed to be independent from other eRCP components, so it is possible to use eSWT without eRCP as part of a "midlet"[9].

There are several technologies for GUIs design for mobile devices, the most popular are the ones derived from the desktop implementations of Swing, AWT and SWT indeed.
Why we chose eSWT? Right, it comes as part of eRCP but our choice is not related to that but rather to usability (once again!) and performances justifications. Let's analyze the competing technologies to prove that.

[9]eSWT has has lighter hardware requirement compared with eRCP, it requires a CLDC JVM (instead of the CDC one) and thus can run as a midlet even on low-end phones

| Component | SWT | Swing | AWT |
|---|---|---|---|
| Button | X | X | X |
| Advanced Button | X | X | |
| Label | X | X | X |
| List | X | X | X |
| Progress Bar | X | X | |
| Sash | X | X | |
| Scale | X | X | |
| Slider | X | X | |
| Text Area | X | X | X |
| Advanced Text Area | X | X | |
| Tree | X | X | |
| Menu | X | X | |
| Tab Folder | X | X | |
| Toolbar | X | X | X |
| Spinner | X | X | |
| Spinner | X | X | |
| Table | X | X | X |
| Advanced Table | X | X | |

**Figure 4.11:** Visual Component Comparison

The AWT (Abstract Windowing Toolkit) framework uses native-looking widgets[10], it come as part of CDC Personal Profile JVMs but unfortunately suffer from a LCD-problem[11]: in a nutshell if a platform A provides ten widgets and platform B has that ten widgets plus twenty more, the cross-platform AWT framework only offers the intersection of these two sets (figure 4.11 and 4.12).

Thereof we have evaluated that AWT framework doesn't supply enough visual components for developing our system GUIs.

Swing is considered the standard Java SE framework for GUIs implementation. It provides a large set of features, comes up with elegant look and feel and results easy to use from a developer view due to a high abstraction level

[10] A Widget is the smallest unit of a UI: for instance windows, buttons, tables, popup windows are all widgets

[11] Lowest Common Denominator problem

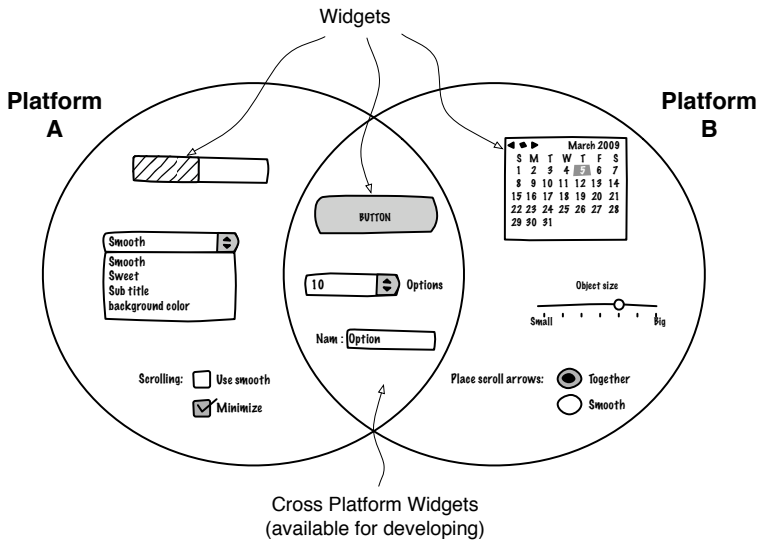



**Figure 4.12:** The AWT LCD Problem

of the implementation concepts. From a technical point-of-view it solves the LCD-problem using emulated widgets instead of natives. This solution entails some important drawbacks that crash against UbiCollab AR-DE-03[12] requirement: Swing application no longer look like native applications, moreover, because widgets have to be emulated, Swing applications consume too much memory thus this technology is not optimal for mobile devices.

eSWT go beyond these issues normally acting as Java wrapper[13] around the operating system's native widget, as AWT does, but emulating widgets, as Swing does, when some of them are not available on the host platform. This approach not only lends eSWT-based application the look and feel of native (non-Java) applications, but also boosts their performances, since native widget libraries are likely optimized for their target operative system.
As downside of this technique eSWT is only supported on platforms for which a platform-specific eSWT implementation exists, but being conscious that in computer science portability and performances are competing issues the Eclipse engineers decided to sacrifices portability across different mobile

[12] AR-DE-03: System's GUI provided for User Interaction has to use OS native widgets
[13] Through JNI - Java Native Interfaces

platforms to achieve enhanced performances and especially usability. This modus operandi is highly compatible with UbiCollab design rules, thus we chose eSWT (coupled with eRCP) in order to implement our GUIs. A proof of goodness of the choice made can be found digging in the eSWT architecture (figure 4.13), three components are included into:

- Core
- Expanded
- Mobile Extension

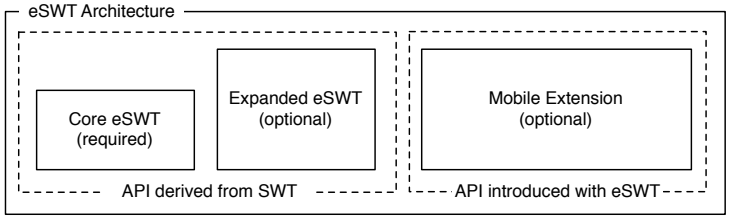



**Figure 4.13:** eSWT Architecture

This kind of componentization allows for flexibility to configure what components should be included in the device, based on device capability and purpose. The core component, mandatory on every distribution, is a subset of basic desktop SWT API, including low-level graphics, events, and basic widget infrastructure. The Expanded component contains a subset of more sophisticated desktop SWT widgets, such as layout managers. These require resources commonly found on high-end mobile devices and PDAs. The Mobile component includes widgets, such as dialogues box and controls, targeted for embedded devices. This component plays a fundamental role in usability providing support to device-proprietary input/output mechanism and it uses the native UI capabilities common to mobile devices to better adjust eRCP applications to different devices. Instead of desktop machines, which all share common features in terms of screen sizes and pointer mechanism, mobile devices come in a wide range of shapes and sizes and have a variety of input mechanisms. As much as possible, a developer involved in Ubicollab

UIs design has to keep on mind that his/her GUIs should run *well* on any kind of mobile device. Usability is more difficult to accomplish for mobile devices where environments vary and expectation for ease of use are very high. eSWT and Mobile Extension attempts to normalize devices so that the application programmer does not have to do a lot of work to handle the differences among devices. It does it in two ways: implicitly, by providing a device's native look and feel that a user is familiar with, and explicitly, by providing mechanism that abstract input and output through the actual device hardware [21].

Implicit normalization is automatically provided since eSWT widgets are implemented using a platform's native widgets, they appear and behave similarly to widgets in native applications. The end-user can recognize and interact with these widgets as his/her is used to.

Explicit normalization is provided via specific mechanisms that a programmer is encouraged to use. These generally fall into two categories: organizing output on a display and handling different input mechanism.

At the present day the Eclipse foundation officially provides eSWT implementations for Windows, Windows Mobile, Symbian OS. There are even unofficial implementation for some linux mobile OS as Jalimo project [22] for Linux Maemo[14], a popular linux distribution shipped with Nokia tablet PCs. A typical invocation to a widget and its own connection with the native implementation is schematically presented in figure 4.14

**eJFace**

eJFace wraps the eSWT widgets in the context of the Model-View-Controller (MVC) paradigm. In short, it assures data binding among model, controller and view classes. It also provides resource-handling classes that allows for efficient manipulation of resources as fonts and images, thus increasing system performances.

[14]Maemo Community - http://maemo.org/

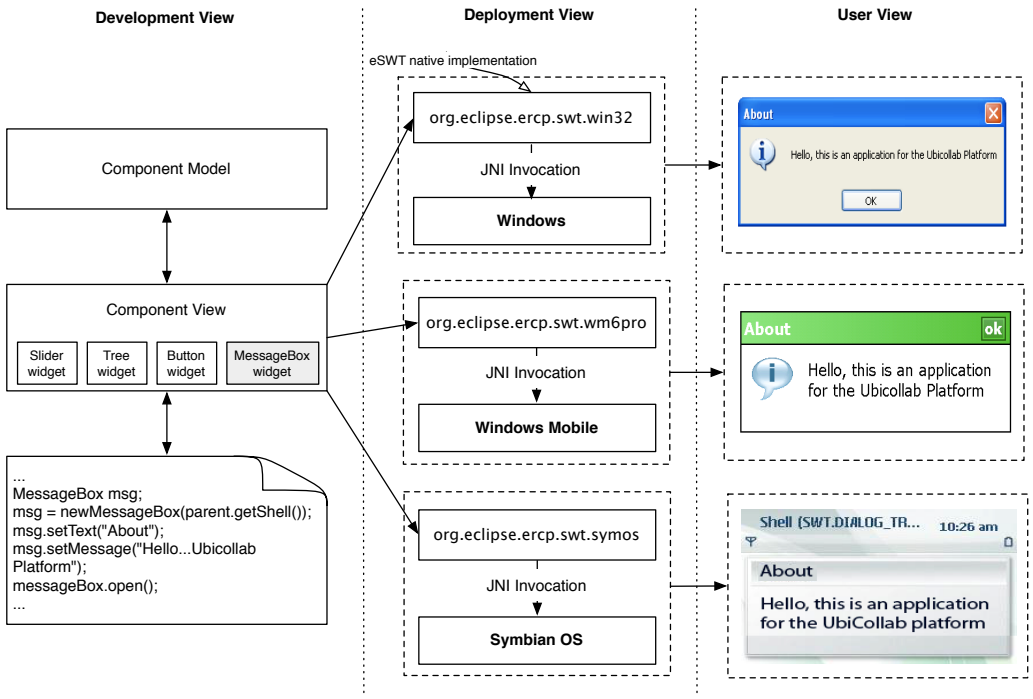



**Figure 4.14:** Native Widget Invocations on different Operative Systems

eJFace provides the mechanism by which plugins programmatically contribute to the workbench, which is further discussed in the next section.

### eUpdate

eUpdate is a bundle part of the eRCP distributions that handle other bundles (plugins) installation and upgrade by the update site mechanism typical of the Eclipse Products[15] This bundle management system is not used in UbiCollab since it comes with a proprietary bundle discovery and management apparatus[16] and thereby eUpdate has been excluded in the UbiCollab customized eRCP distribution

### eWorkbench

eWorkbench provides an implementation of concepts that came in the picture

[15]Developers which use the Eclipse IDE are familiar with this plugin installation modality

[16]The Resource Discovery Manager, SD Plugins and Service Domain Manager bundles

in Chapter 3. It is included in eRCP framework as a draft[17] component that can be expanded and customized according with the Eclipse EPL license [23]. Even if is a relative new technology, and customizing it is not an easy task, it has become a central component of several successful commercial product like the IBM WebSphere Everyplace Micro Environment[18] and the Sprint Titan Framework[19] due to its high integration with eSWT, which is the component that renders the GUIs, and with OSGi, which guarantee integration with other bundles and Service Oriented approach. Our implementation of the eWorkbench draft will be presented in the next section.

## 4.2 Components Architecture

Thanks to its inner modularity an Ubicollab distribution can be customized and selectively composed, saving resources, in plenty of different configurations addressing specific scenarios needs. A distribution is built with *components* which share the same internal architecture. A component is arranged in mandatory and optional *units*, as depicted in figure 4.15.

A component is made up of the following units, sorted in mandatory and optional:

- **manifest.mf** file [mandatory]: it contains informations required to connect the component to the OSGi framework and properly run it, like: classpath, imported and exported packages.

- **plugin.xml** file [optional]: it contains informations required to connect the component to the eRCP framework, like number and names of views and their implementation paths; it is required just if the bundle provide user interfaces.

[17]Sources of the draft component are available from Eclipse cvs: http://dev.eclipse.org/viewcvs/index.cgi/org.eclipse.ercp/core/

[18]IBM WEME: http://www-01.ibm.com/software/wireless/weme/

[19]Sprint Titan Framework: http://developer.sprint.com

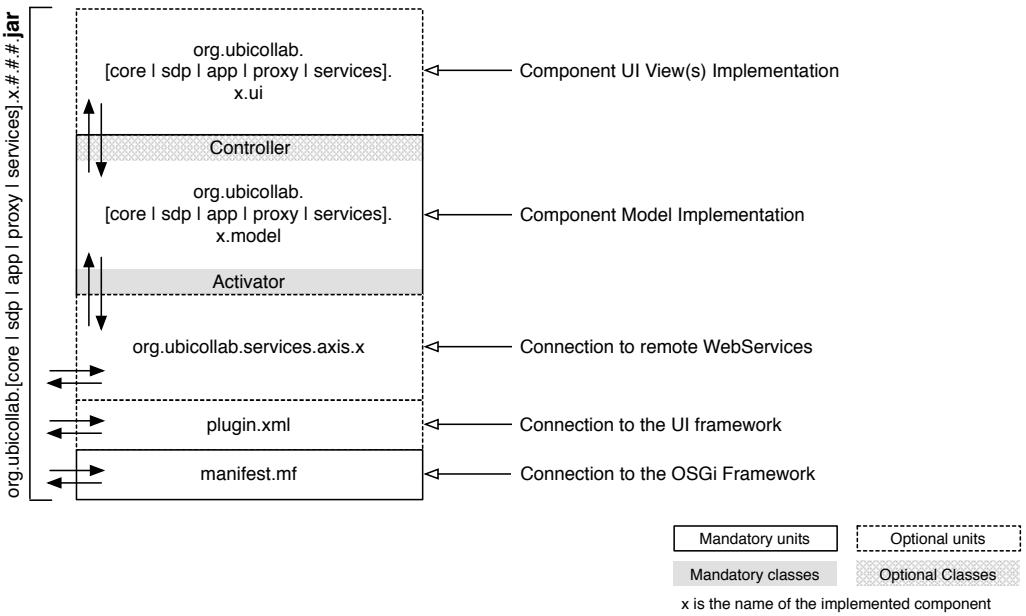



**Figure 4.15:** Component's Internal Architecture

- **org.ubicollab.services.axis.x** package [optional]: it contains classes needed in order to create a local stub of the webservice x and thus let the bundle call remote procedures on it. This package is generated at compile time by the Apache Axis[20] engine fed by the wsdl file of the service we want to connect to. It is an optional unit since is only included in component that necessitate to directly communicate outside the UbiCollab framework, as proxies for instance.

- **org.ubicollab.[core / sdp / app / proxy / services].x.model** package [mandatory]: this package encloses all the internal procedures and algorithms that implement bundle x functionalities. The model includes the activator class which is used to wire the bundle with the OSGi framework and thus get services references and system events notifications. If the bundle provide UIs is a good practice to also have a controller class used to drive UIs in accordance with the model-view-

[20] Apache Axis: http://ws.apache.org/axis/

controller pattern[21]. Due to the high complexity of this package is allowed and suggested to organize it in sub-packages.

- **org.ubicollab.[core / sdp / app / proxy / services].x.ui** package [optional]: it includes classes which implement user interface functionality, as well as GUIs or other user interaction mechanism. It has to be designed using interfaces provided by the underlying model package and thus can be modified and more user interaction can me added afterwards without re-implementing the model. This package may have dependencies with the UIToolkit bundle wich feeds the GUIs with buttons and other graphical widgets.

All these units are deployed in a versioned jar file named *org.ubicollab.[core / sdp / app / proxy / services].x.#.#.#.jar* where x the represent component name and #.#.# the component version. Besides device embedded service implementation, the only exception to this set of rules is represented by the eWorkbench component since, as it will be explained latter, it act as link between the eRCP framework UC components; therefore has to adhere to both framework specifications.

## 4.3 Components Implemented

### 4.3.1 Implementation Overview

All the components implemented as contributions to the research work are schematically presented in figure 4.16. All the components run on the top of the implementation stack reported in figure 4.17.

Each component implementation is analyzed in the following sections.

[21]The jFace toolkit included with eRCP can be used to simply pattern implementation

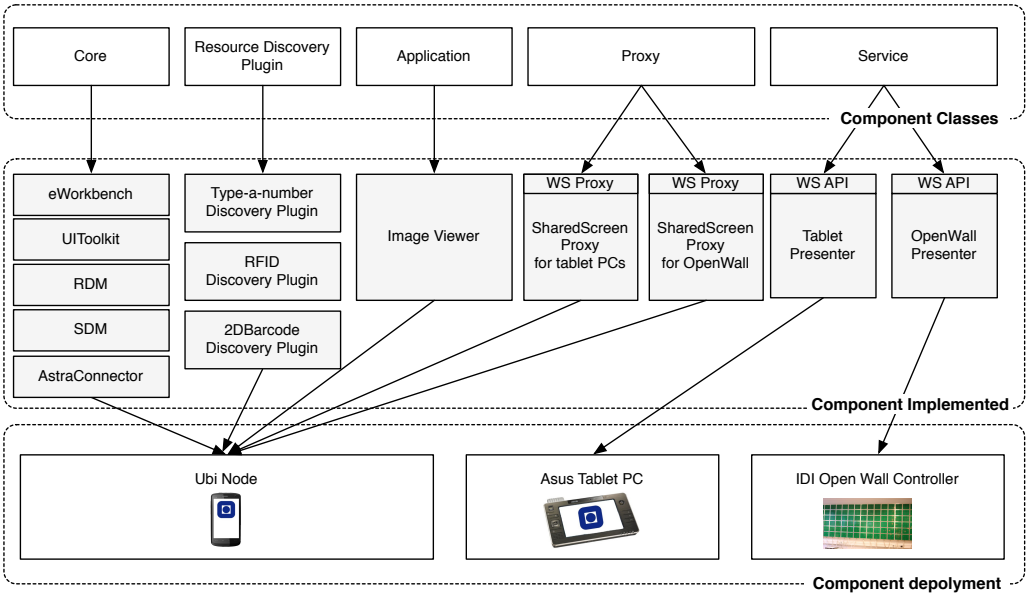



**Figure 4.16:** Implemented Components Overview

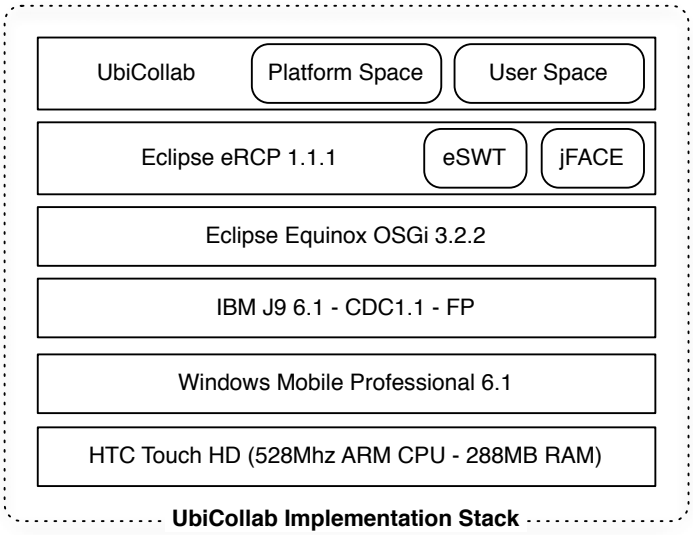



**Figure 4.17:** UbiCollab Implementation Stack

### 4.3.2 Platform Components Enhancements

Because the modules implemented as contribution of this work rely on Resource Discovery and Service Manager modules developed in a previous research work [2] these have been updated to be compatible with the new project specifications and running environment. Compilation units have been renamed according with the conventions reported in Section 4.2. Since before the current work UbiCollab was running over Knopflerfish OSGi implementation, the manifest file has been rewritten to fulfill equinox specifications, that are more constraining compared with Knopflerfish's in terms of bundle's activation policies and dependencies management. Thus the version of these components have been moved from 0.5 to 0.6 for the Service Domain Manager and from 1.0 to 1.1 for the Resource Discovery Manager (which was named Service Domain Manager according with the out-of-date specifications). No reengineering of the source code have been made since it was already suited for the requirement introduced in mobility area.

### 4.3.3 eWorkbench

Building on the Eclipse Foundation draft we developed a customized eWorkbench. Modification has been involved the reimplementation of the perspective concept, in order to fit the UC design, and the customization of the GUI layout and look and feel in order to accomplish UbiCollab requirement and produce an implementation of mockups presented in Chapter 3. Now we are going to explain how we have interpreted the eWorkbench idea, starting with some general concepts that the reader may don't know if is not familiar with Eclipse products designs.

Generally speaking the eWorkbench supply a technology to create a visual framework for displaying plugins[22] UIs and, thanks to the OSGi layer, allows

[22]In eRCP terminology a plugin is any OSGi bundle that provides one or more views and the mechanism to wrap these GUIs in the context of the eWorkbench. Hence, in

them run simultaneously inside a single workbench window. eWorkbench works providing *extension points* that the plugins extends. An extension point is the definition of a port, an entry-plug for other plugins to offer services. It could be better understood if we consider an extension point similar to a Java interface. Like an interface, an extension point defines a contract between the user and the service provider [24]. In our implementation it offer to other UbiCollab components to contribute to User Interactions provided by eWorkbench publishing one or more views in a determinate perspective, all the perspectives belongs to the UbiCollab eWorkbench (figure 4.18); therefore in UbiCollab domain the **service provider** involved in the contract, is an *User Interface Service Provider* impersonated by the eWorkbench and **users** are UbiCollab modules that issue one or more user interfaces to it.

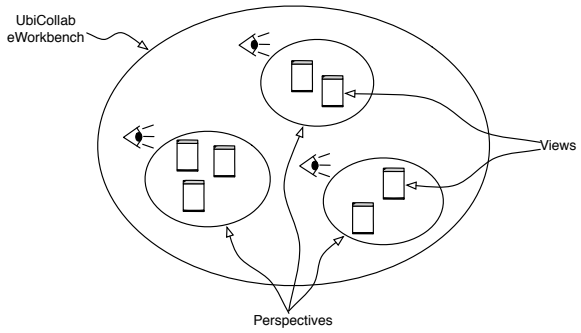



**Figure 4.18:** Perspectives inside a eWorkbench

The contract that binds views and perspectives with the eWorkbench is written in XML language and enveloped in a *plugin.xml* (figure 4.19) file where extension point are defined as well as extension implemented by the plugin.

Still referencing to Java concepts we can assume the extension point as a Java Interface and the implemented extension as a Class that implements that interface. The plugin.xml provides the path where extensions are implemented as well as fingerprints of the extension-points provided. Extension point definitions are enveloped in a XML Schema file with extension `.exsd`,

UbiCollab terminology could be any component that come up with a UI like a Service Discovery Plugin as well as a third-party UbiCollab Application

plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin>

<extension-point id="applications" name="UC eWorkbench"
 schema="schema/applications.exsd"/>

<extension id="eWorkbench"
point="org.eclipse.core.runtime.applications">
<application>
<run class="org.eclipse.ercp.eworkbench.eWorkbench">
</run>
</application>
</extension>
.........
</plugin>
```

Extensions Point Published

Extensions Implemented

**Figure 4.19:** plugin.xml from eWorkbench bundle

thus bundles that want to implement the extension points have to adhere to that schema (figure 4.20).

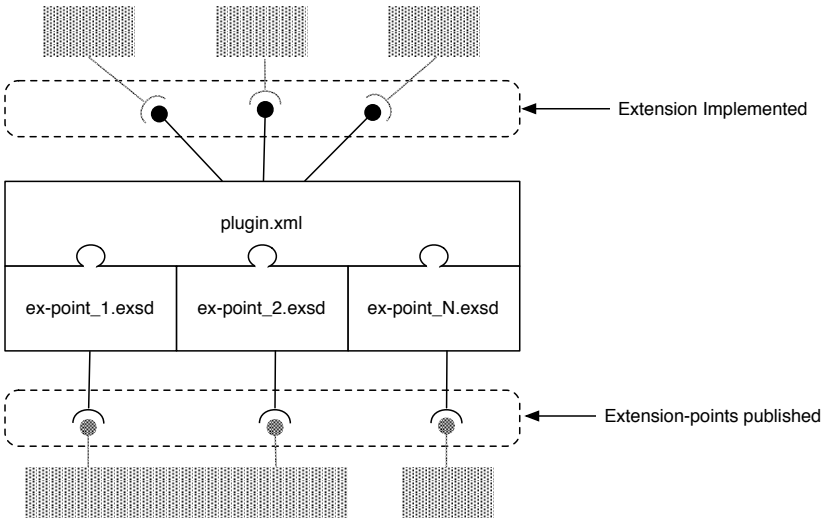



**Figure 4.20:** Extension system for a bundle into the eRCP framework

To be of any use, extension points and extension implementations must find each other. The eRCP framework maintain an extension registry for this purpose, and allows bundles implementing extension to plugin the respective extension point at runtime. It means that a proxy, for instance, as soon is discovered and downloaded is just ready to present its user interface, without

any further setup or reboot operation. UbiCollab eWorkbench offers extension points for bundles that want to publish UIs and implement extension provided by framework core component in order to have low-level access to device displays and other hardware needed to implement view and perspective concepts.

The extension point for UbiCollab components provided by eWorkbench included as part of plugin.xml file is shown in figure 4.21

```
<plugin>                                                                    (1)  (3)
<extension-point id="applications" name="UC eWorkbench" schema="schema/applications.exsd"/>
</plugin>                                                                   (2)
```

**Figure 4.21:** Extension Point Declaration

The `extension-point` tag requires three parameters:

1. **id** is the extension point identifier. The eRCP framework concatenates it with the plugin id to make a platform-wide unique identifier.
2. **name** is a user-friendly name
3. **schema** points to an XML Schema that describes the markup for the extension, stated in figure 4.22

Any UC component that comes with UI mechanism has to use this markup schema in its own *plugin.xml* file.

To make this idea better clear we now analyze how this binding contract is implemented on the user side[23], looking inside the *plugin.xml* file included in the Type-a-number RDPlugin bundle presented in Chapter 3, which code are listed in figure 4.23

A *view* implemented by the component is stored in a Java Package embedded in the bundle and hence deployed in a Jar file. It contains at least a GUI

[23]In this context an user is a bundle that publish UIs in the eWorkbench

3 applications.exsd

```
<?xml version='1.0' encoding='UTF-8'?>
<schema targetNamespace="org.eclipse.ui.workbench">
<annotation>
<appInfo>
<meta.schema plugin="org.eclipse.ui.workbench" id="applications" name="UbiCollab Applications"/>
</appInfo>
<documentation>
This extension point allows bundles to register UIs to the UbiCollab eWorkbench
</documentation>
</annotation>
.....
<attribute name="UCPerspective" type="string" use="required">
<annotation>
<documentation>
Define to which UbiCollab Perspective the plugged module belongs to.
</documentation>
</annotation>
</attribute>
</complexType>
</element>
.....
<element name="views">
<annotation>
<documentation>
The views the plugged module implements
</documentation>
</annotation>
<complexType>
<sequence>
</sequence>
<attribute name="normal" type="string" use="required">
<annotation>
<documentation>
Identifier of Normal view
</documentation>
</annotation>
</attribute>
<attribute name="large" type="string">
<annotation>
<documentation>
Identifier of a alternative view
</documentation>
</annotation>
</attribute>
......
</schema>
```

**Figure 4.22:** application.exsd

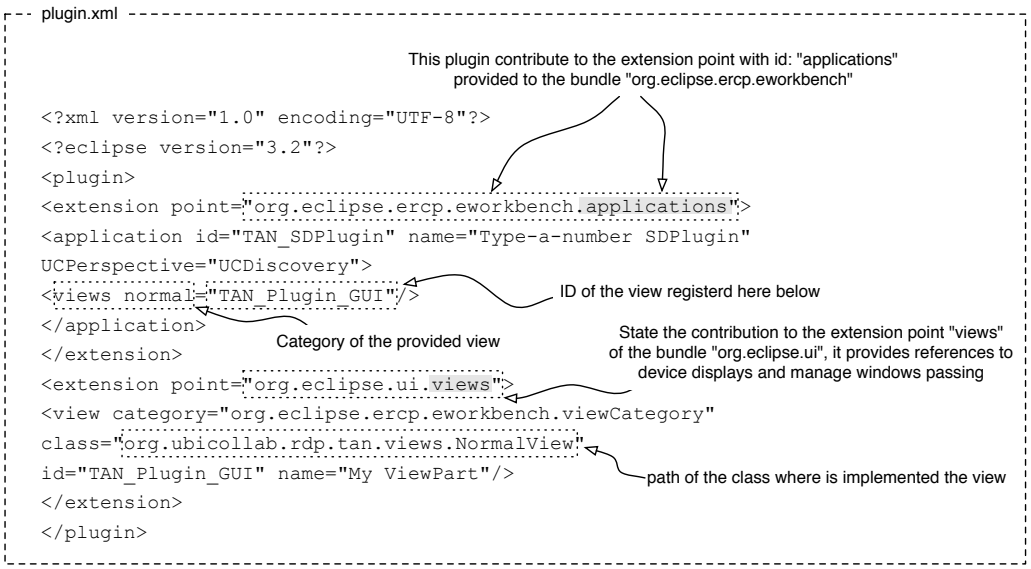



**Figure 4.23:** plugin.xml from `org.ubicollab.rdp.tan` bundle

designed with eSWT and eJFace tools, but that can be enhanced adding additional User Interaction mechanism such as voice or gesture recognition designed by the developer itself or supplied in third-party libraries[24]. For plugins that provides views for specific display scenarios, eWorkbench automatically decides which view has to be prompted based on the hardware capability of the mobile device in use. For example, if a device has two displays, a eWorkbench can display a different application on each display or transfer an application's view from one display to another. When a device is opened, an application can move from a small external display to a larger internal one.

At this stage of the report we can render a more detailed description of the platform abstraction layer and the user abstraction layer reported in Chapter 3 for the UbiCollab eWorkbench component, as reported in figure 4.24 and 4.25.

[24]Third-party user interaction engines can me embedded in the view's bundle or come as separate bundle shared among multiple views.

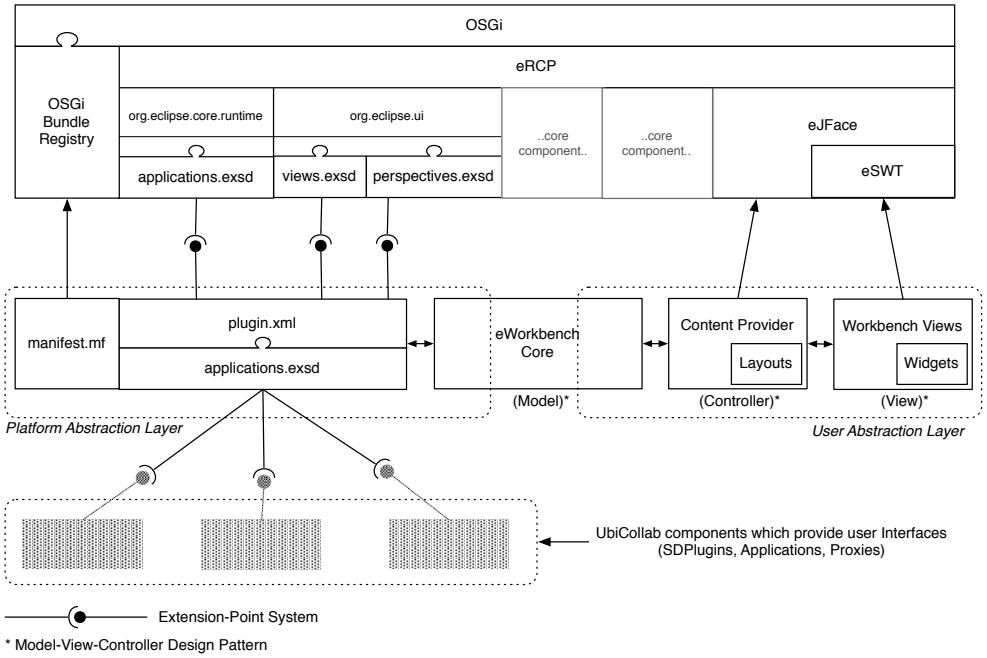



**Figure 4.24:** eWorkbench Architecture

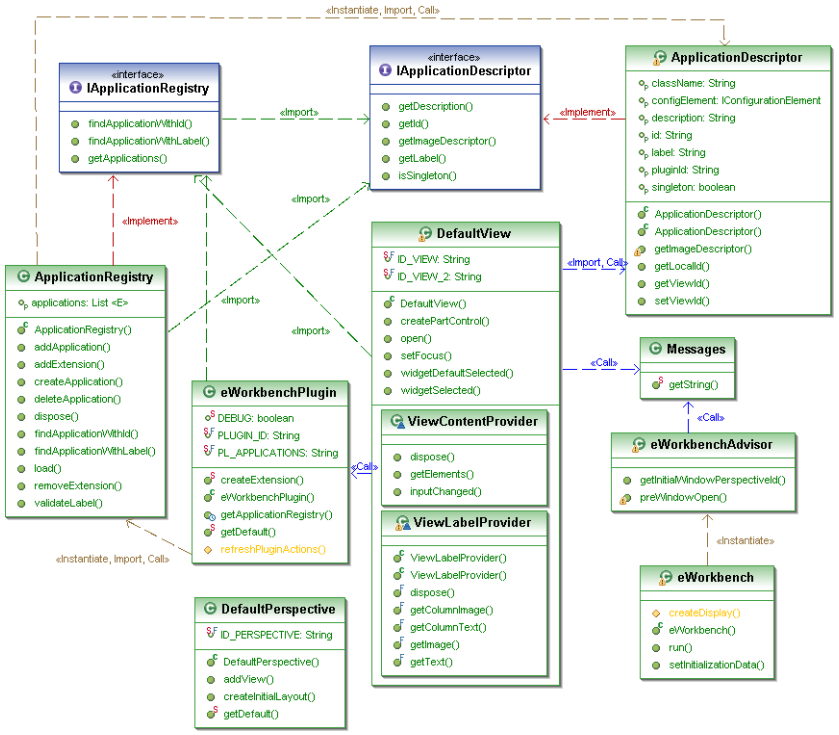



**Figure 4.25:** eWorkbench UML Class Diagram

Finally we can see how the the perspective concept is rendered to the end user: in figure 4.26 are reported some screenshots taken from the UbiNode where is represented the three UbiCollab perspectives with some applications and Resource Discovery plugins installed in the framework.

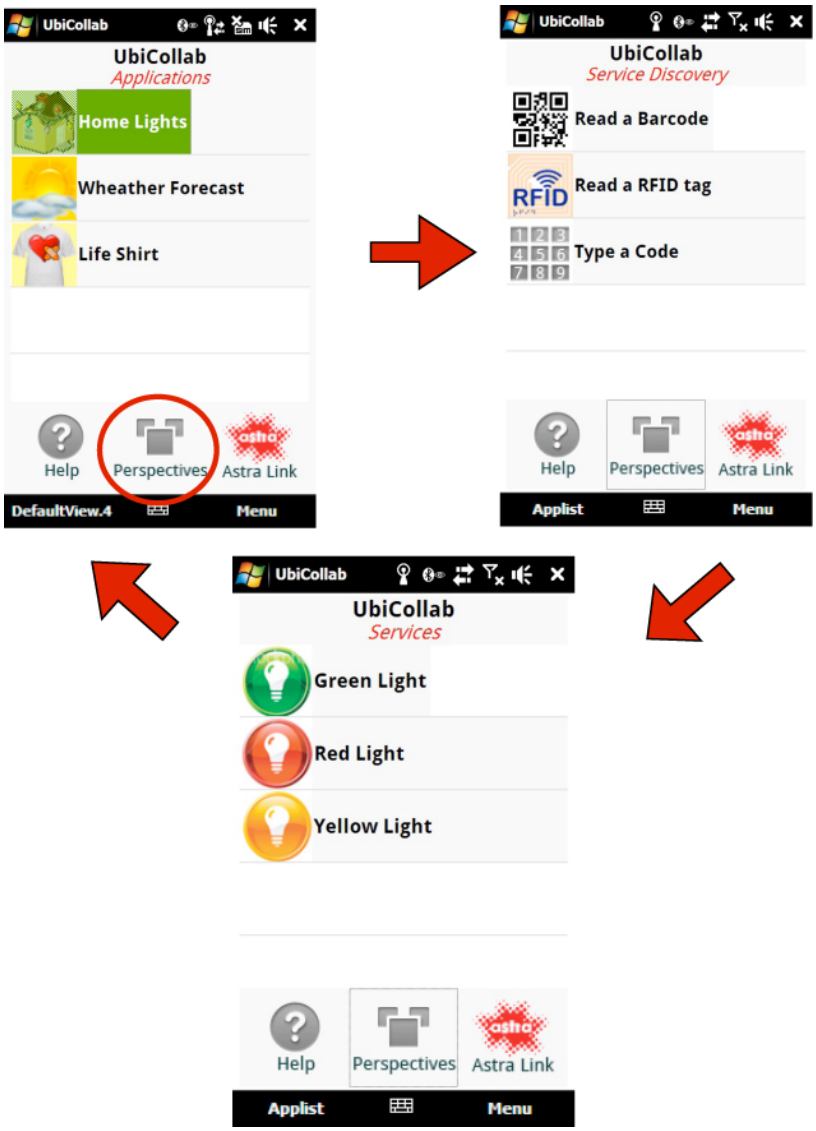



**Figure 4.26:** eWorkbench Screenshots

### 4.3.4 Type-a-Number Resource Discovery Plugin


**Figure 4.27:** A Screenshots from the Type-a-Number Resource Discovery Plugin

The Type-a-Number (TAN) is plugin for the Resource Discovery Subsystem developed by KSJ [2] and make use of the API provided with it. The TAN Plugin implements the *Type a number discovery gesture*, providing to the user an easy way for discovering and adding services to his/her UbiNode. Even if it needs to have the resource on sight and use fingers to digit the number on the PDA screen, as already pointed out, it work well in a lot of scenarios; moreover due to its low hardware requirement it can be deployed as part as every UC distribution.
In short the plugin resolves the number entered by the user looking for a matching Service Advertisement.
The user can find the numeric code which identify a resource on a label or screen located close to the resource. Next to the numeric code can be also reported other *Discovery Gestures* available for that resource. A typical advertisement is shown in figure 4.28, this advertisement links the the user domain with the underlying *Service Advertisement*

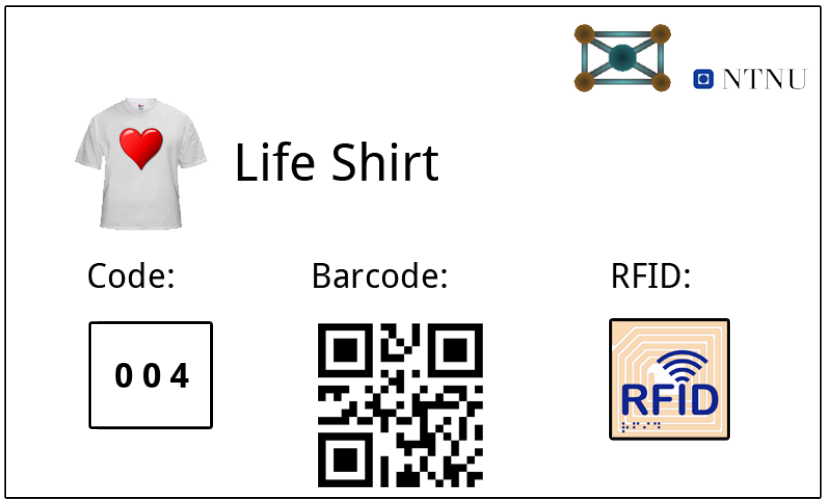



**Figure 4.28:** Advertisement for an UbiCollab Resource (a prototype of a Biomedical Shirt )

The Service Advertisement (SA) is one of the most central concepts for Resource Discovering. SA is the descriptive information concerning a service that is made available for potential *service requesters* [2] .

A Service Advertisement includes these fields:

- Service id: the 4 digits number which identify the resource
- Name: an user-friendly name for the resource, e.g. "Tablet PC".
- Location: the location where the resource resides, e.g. "IT-Bygg, room 054"
- Owner: the owner of the shared resource, e.g. "NTNU IDI Department"
- Type: The service type. Can be used by applications to determine how to handle a discovered service.
- Description: a short description of what the service does
- ServiceUri: the URI which point to the resource proxy jar file, needed in order to communicate with the WebService implemented by the re-

source

- DescriptionUri: a URL which point to a an HTML page with informations about the resource and how to use it.

When a user initiate a Service Discovery gesture, typing a name in this instance, the intrinsic information embodied in the gesture is matched against available service advertisements. Since a resource is listed by a 4 digits number spacing from 0 to 9, we can index up to $10^4 = 10.000$ resources. Because UbiCollab seek to be platform independent, XML is used to describe services, in this way service informations will be used and interchanged among several platform components. Thereby each SA has to be validated against the XML schema listed in figure 4.29.

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

**Figure 4.29:** Service Advertisement Schema

Using XML has even the benefit that the SA list can be stored in a webserver and kept updated whereas a new resource would be made available or adapted to work with UbiCollab.

The design of the component is reported in figure 4.30: the plugin at startup time tries to retrieve an updated version of the Service Advertisement list from the UbiCollab server, if it cannot connect to the server (network failure or server down) it search for a cached version of the list in the user space.

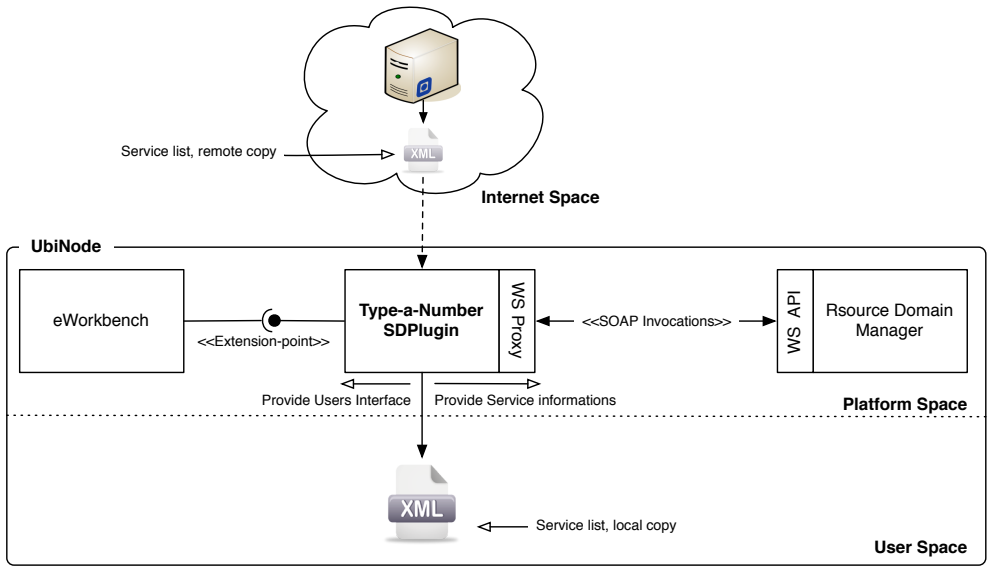



**Figure 4.30:** Architectural view of the TAN Plugin

If the plugin cannot have use of the network and the cached list both (file corrupted or nonexistent) the plugin recover the service list file from a copy included with the plugin when was released, and store it in the user space, ready to be updated as soon as the connection to the server will be available again.

The discovery and service install operations generated by the end-user discovery gesture involves several classes as shown in the sequence diagram reported in figure 4.31.
The component Class diagram is also reported in figure 4.32. Note that the Resource Discovery Manager and the Service domain manager are classes automatically generated from the wsdl document exposed by the those modules. In our framework they act a stubs[25] reflecting all methods invocation to the RDM implementation via the SOAP protocol. Thereby the TAN plugin can be also deployed on a device where RDM and SDM are running

[25]In the distributed computing environment, stub stands for client side object participating in the distributed object communication. It acts as a gateway for client side objects and all outgoing requests to server side objects that are routed through it.

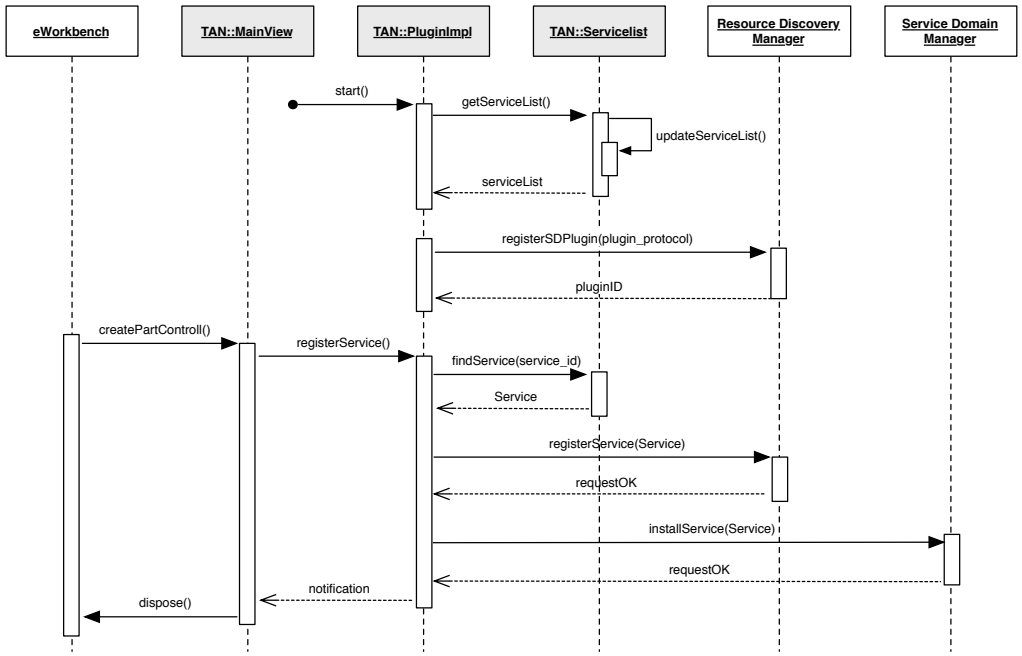



**Figure 4.31:** Sequence Diagram for Resource Discovery Operations

remotely. This feature allows distributed service discovery scenarios since more discovery plugins can be distributed on multiple UbiNodes registering the discovered services within centralized RDM/SDM modules. We can think to barcodes based inventory or goods scanning as scenarios for this strategy.

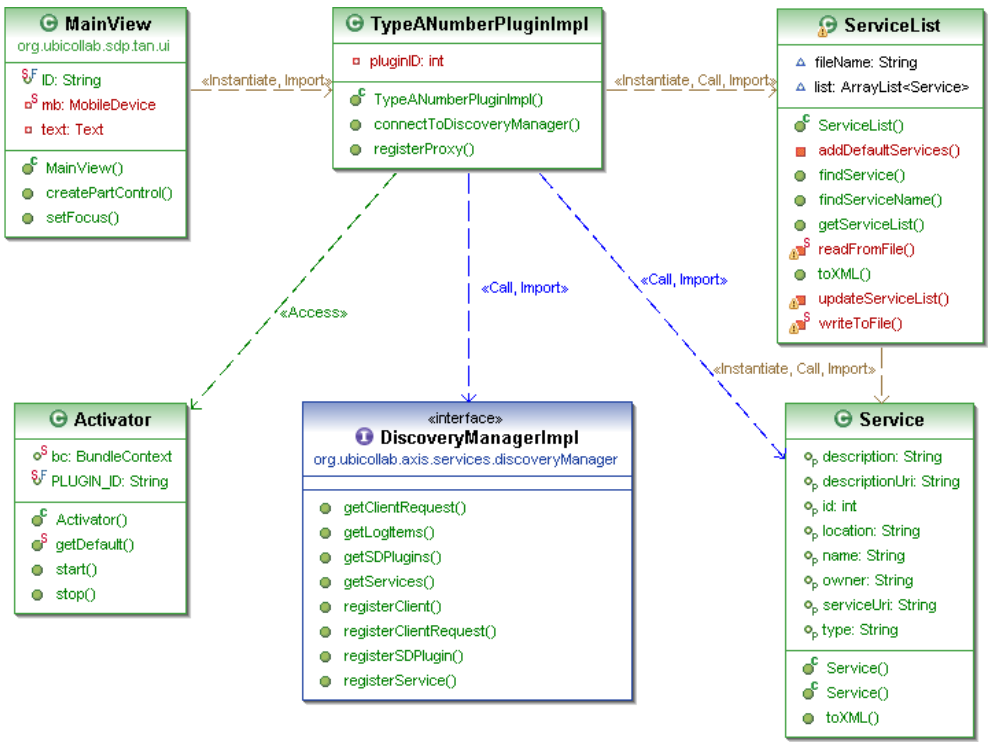



**Figure 4.32:** Class diagram for the Type-a-Number Resource Discovery Plugin

### 4.3.5 RFID Resource Discovery Plugin

The RFID RD Plugin is another plugin for the Resource Discovery Subsystem. Like the Type-a-Number plugin it feeds the Resource Discovery Module with a Service Advertisement, but in this case the tag contains an URI which points to an XML-file which wrap the Service Advertisement. RFID tags (figure 4.33) are small enough to be hidden in a paper label or in a electronic device.

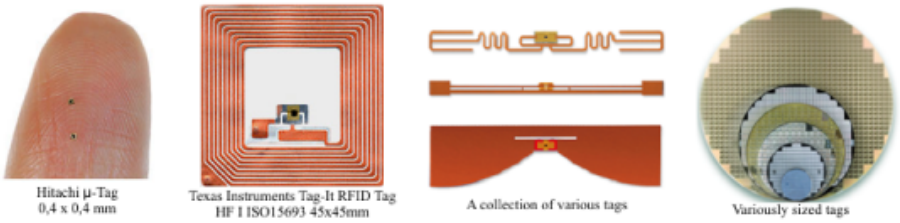



**Figure 4.33:** A collection of different RFID tags.

Since the smartphone hosting the UbiNode doesn't have an embedded RFID reader we had to use an external one. The reader selected for use with the resource discovery plugin is named ID-Blue and is produced by Cathexis Innovations[26]. This is a RFID reader in the shape of a pen. An overview of the technical details of interest for this device is available in Appendix C. This reader communicates with the tag on the proprietary RFID frequency and with the UbiNode by the Bluetooth protocol, allowing the user to discover a resource just pointing the RFID pen to the advertisement tag (figure 4.34) from a distance up to 4cm.

The IDBlue reader supports three different different RFID standards, and in this solution the RFID tags selected for use are ISO 15693-2 compliant tags. Tags offers 2048 bits of storage space, which equals 256 bytes or characters provided if an 8 bit character set is used. This encoding scheme provides the characters necessary to encode almost any legal URI, which is what will be encoded on the tags; anyway if this string lenght is too short for some uses, an

[26]http://www.cathexis.com/

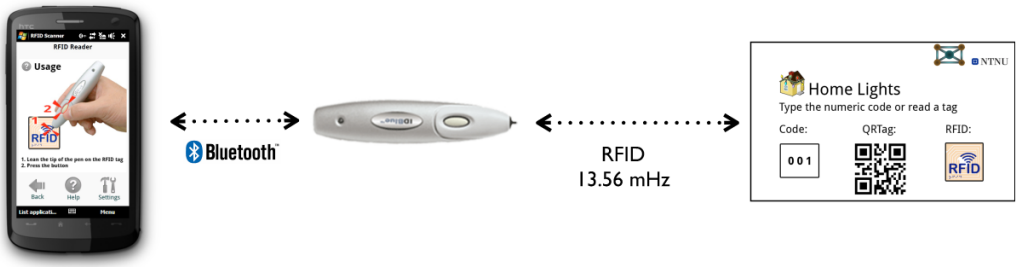



**Figure 4.34:** UbiNode - RFID Interaction

intermediate should be utilized for redirecting. An URI redirecting is simple to make and in addition there are many free solutions for this available on the Internet. One possibility could be to use a service like TinyURL.com[27] to map a short URI e.g. http://tinyurl.com/xyz to the real URI which can have an arbitrary length. The URI encoded in the tag must point to an XML-file with service advertisement. There also exists another option for tagging which does not require the encoding of a URI. This possibility relies on the existence of a SAs Database, and will map the unique tag serial number (stored in each tag) to one service advertisement in the database.

The Bluetooth communication with the reader is established by the open source *BlueCove*[28] library, which implements the JSR-82 and provides API for discovering and communicating with Bluetooth devices. This library has been embedded in the plugin OSGi bundle.

Sequence Diagram for the discovery procedure is reported in figure 4.35

[27] http://tinyurl.com
[28] http://bluecove.org

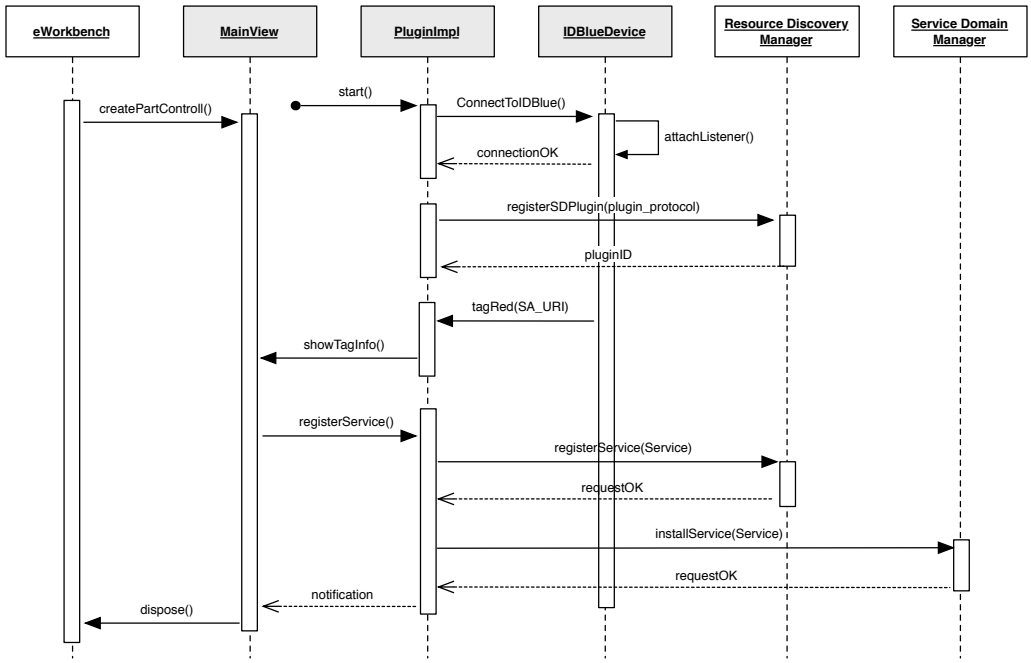



**Figure 4.35:** Sequence Diagram for the RFID Discovery

### 4.3.6 2DBarcode Resource Discovery Plugin

The 2DBarcode RD Plugin is a plugin for the Resource Discovery Subsystem. As we well know it feeds the Resource Discovery Module with a Service Advertisement, but in this case the tag hold an URI which points to an XML-file with the Service Advertisement. The URI is encoded in a two-dimensional barcode. It allows the user to discover a resource taking a photo (with the camera embedded in the smartphone) of a 2DBarcode.

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

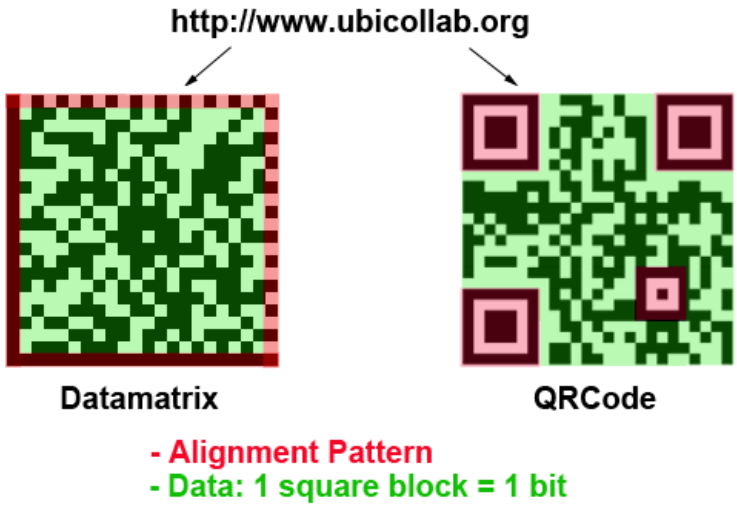



**Figure 4.36:** Datamatrix and QuickRenspose Barcodes

There exists several 2D barcodes which are created for use with camera phones. Perhaps the most interesting are the Quick Response Code (QR Code) and the Datamatrix (figure 4.36), both supported by the discovery plugin.

These two standards exhibits several interesting features:

- **Support for omni-directional (360 degrees) decoding.** This mean you do not need to worry about the orientation of camera reading the barcode.
- **Higher Data Storage Capability (compared with 1DBarcodes).** Tags can store up to 2Kb of informations
- **Advanced error correction capability.** The Reed-Solomon[29] algorithm allows decoding when up to 30% of the code to be damaged.

Limitations of the QR Code include distance and direction constraints, and also the requirement of a high quality camera.

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

The Barcode decoding requires the following steps:

1. The plugin uses JNI[30] to access, via the `Camera.dll`, the phone native camera application and takes a photo with it.

2. The taken image is processed by the `SWTMonochromeBitmapSource` class which uses the `ImageData` class provided by eSWT to extract the RGB[31] value from each image pixel. Since we don't need pixels' color informations but rather we're interested to estimate if a pixel is black or white, the RGB values are weighted by coefficients and summed (equation 4.1). The returned value estimates the *luminance* of each pixel and is matched against a *"black threshold"* to guess if a pixel is black or white.

$$luminance = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{4.1}$$

3. The luminance values are stored in a matrix whose size is `[image.weight x image.height]`. Now the informations wrapped in the barcode are ready to be decrypted.
   The barcode decoding algorithms are implemented by the ZXing[32] library, an open-source, multi-format, barcode image processing library. The library has been recompiled to adhere to the specification of a CDC FP JVM and as been deployed in a OSGi bundle which exposes a barcode decoding *service* to other bundles.

4. Finally the discovery plugin invokes the barcode decoding service sending the *luminance matrix* via the OSGi service interface. The service, in absence of exceptions, match the luminance pixel matrix against

[30] Java Native Interface, is a framework that allows Java code running in a Java Virtual Machine (JVM) to call and to be called by native applications (programs specific to a hardware and operating system platform) and libraries written in other languages, such as C, C++ and assembly

[31] RGB is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors

[32] http://code.google.com/p/zxing/

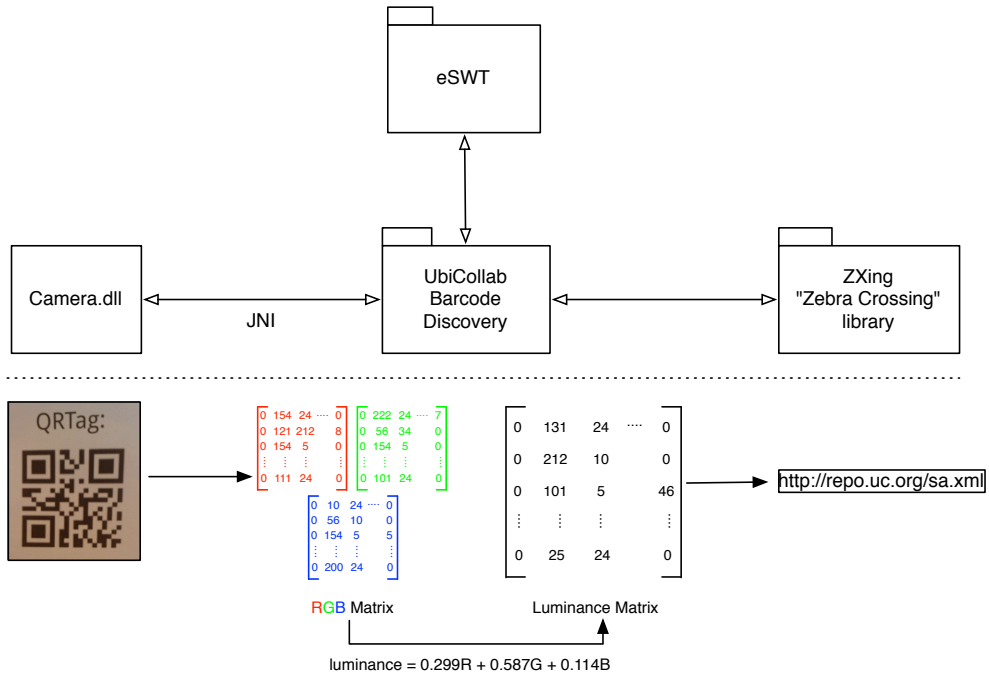



**Figure 4.37:** Barcodes decoding procedure

code patterns returning a string containing the URI of the service advertisement.

The full decoding procedure is illustrated in figure 4.37

### 4.3.7 AstraConnector

The analysis of the *AstraConnector* module implementation would be easier to understand separating the two side of the communication between the platforms: the UbiCollab to ASTRA connection and the ASTRA to UbiCollab connection.

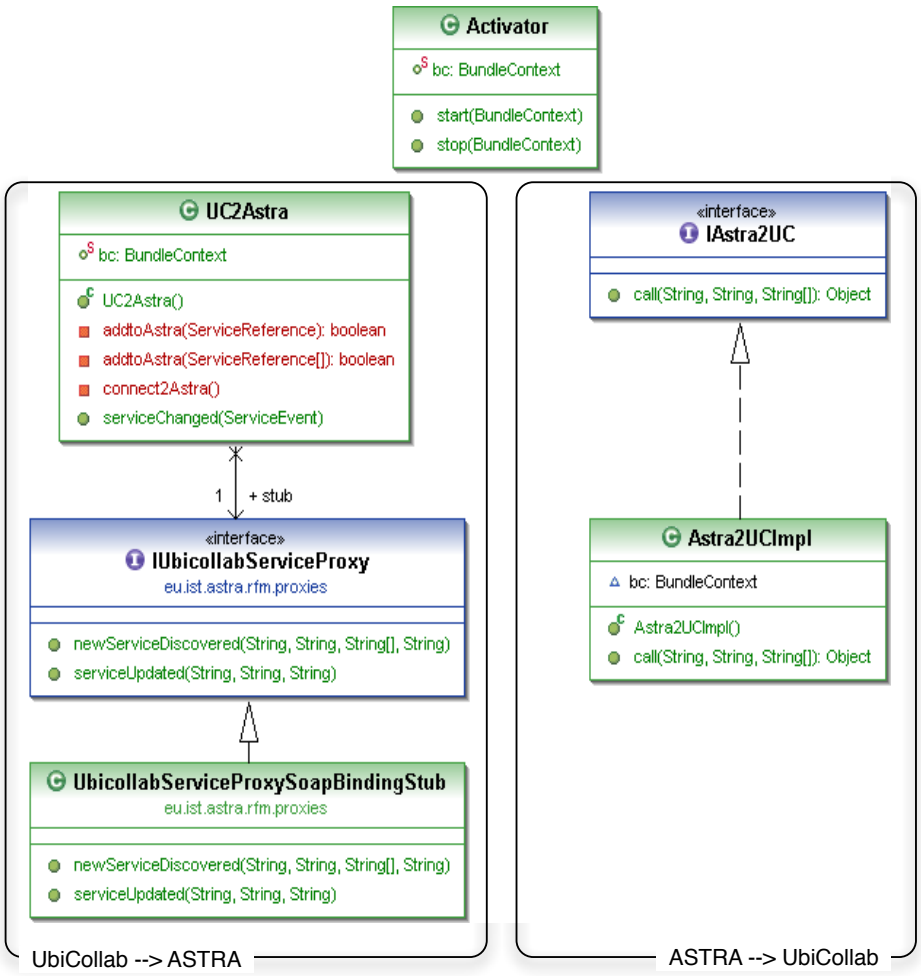



**Figure 4.38:** UML Class diagram for the AstraConnector module

### Exporting services from UbiCollab to ASTRA

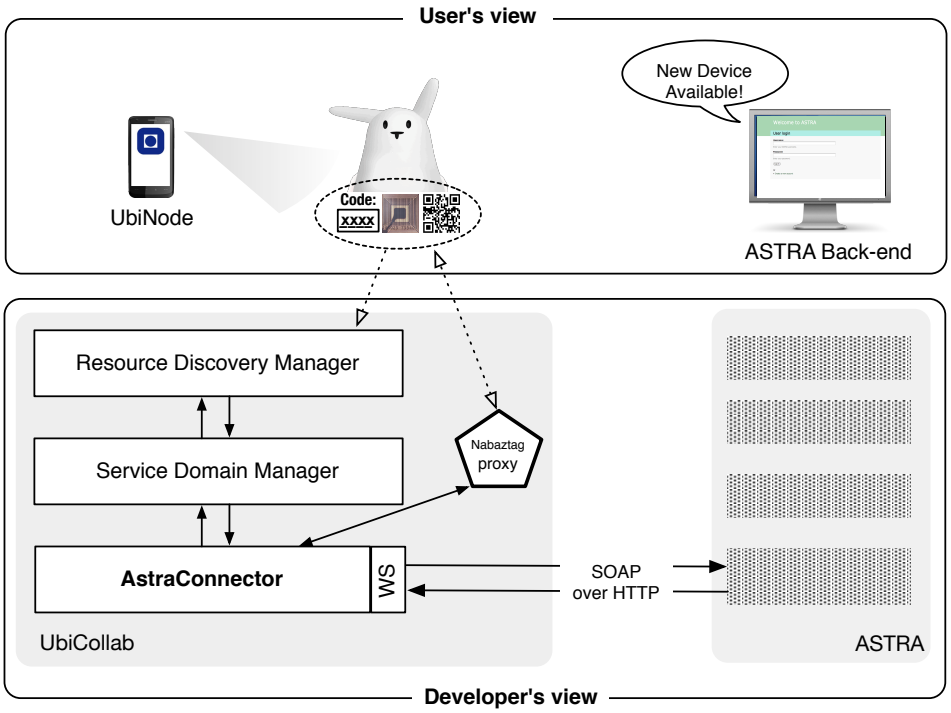



**Figure 4.39:** User vs. Developer point of view of the integrations

The Service (resource) export task involves the following steps:

1. UbiCollab connects to Astra via WebService
2. The UC2Astra class connects to the Resource Domain Manager and retrieve the Service References of the proxies installed in the UbiNode
3. The system scans the Interface exposed by each proxy and converts them in a textual notation ready to be sent to Astra via SOAP invocations
4. UbiCollab register each proxy with Astra providing the name of the resource and the actions (methods) that the resource is able to perform.

Moreover the AstraConnector is registered as listener on the OSGi framework is therefore able to fetch real-time notification about resources discovered by

the user after the system startup and export them to ASTRA.
The overall procedure is highlighted in the sequence diagram in figure 4.40

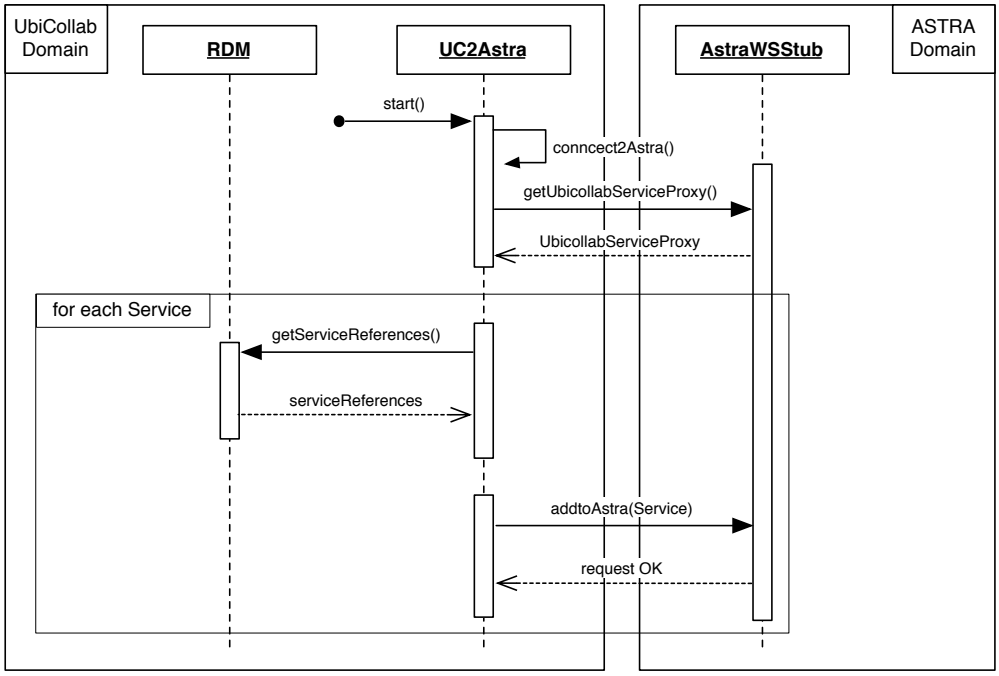



**Figure 4.40:** Sequence diagram for Service Proxy export to ASTRA

### Using UbiCollab Resources Proxies from ASTRA

UbiCollab allows remote service invocation from ASTRA by exposing a Web-Service interface dedicated for it. The Interface includes the metod `call()` (figure 4.41) that require as parameter the name of the Proxy service, the name of method defined in the interface of that proxy and an array filled with the parameters that the chosen method requires. The AstraConnector

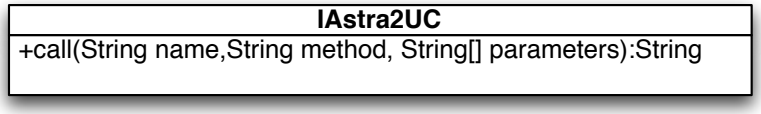



**Figure 4.41:** WebService Interface exposed by the AstraConnector

exploits the power of Java Reflection[33] to build real time instances of the proxy implementation and call methods on them.

Reassuming, methods invocations are generated inside ASTRA by an application, sent to UC via WebService, routed by the AstraConnector to the correct real time proxy instance and finally sent to the final device by the proxy itself. In the example shown in figure 4.42 we can see how the *AstraConnector* works: it get from ASTRA the invocation `call(nabaztag,rotateHear,90)`, it look in the service registry for a proxy called "nabaztag" and build, thanks its interface, an instance of it; finally it call the method `rotateHear(90)` on the nabaztag proxy. Then the proxy converts the method invocation according with the proprietary interface running on the remote resource, `hear_pos(5A)` and send it.
Finally the resource, the nabaztag, performs the selected actions: it rotate 90deg its hears.

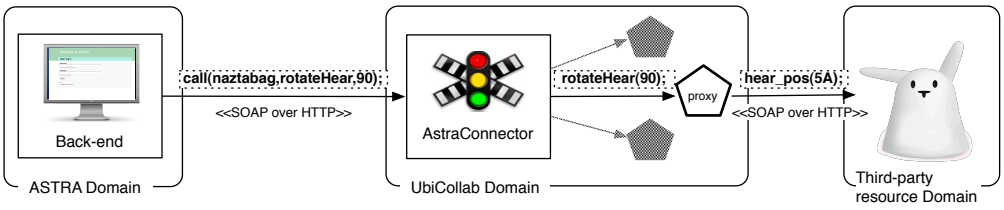



**Figure 4.42:** Procedures invocations from an Astra Application to a UbiCollab Resource

[33]For more datails about Java Reflection visit:
http://java.sun.com/j2se/1.5.0/docs/api/java/lang/reflect/package-summary.html

### 4.3.8 ImageViewer Application

The ImageViewer app is an example of a composite application that make use of several UbiCollab concepts: application, proxies and services.
Basically is a software developed for the UbiNode, resource comply for mobile devices indeed, that shows pictures downloaded from a remote user spaces like a shared folder on a webserver. It response to screen touches browsing through the digital contents and showing them in a full-screen modality. But it is not all this. ImageViewer can connect to external shared screens -a shared resource in UC terminology- and drive them showing an adapted version of the contents the user is watching on the PDA, as an high-res version of a photo for a big size shared screen or a descriptive text on a shared led screen. In order to run the application in this distributed modality the user has to discover and install proper proxy, called SharedScreen Proxy, the discovery gesture required by the discovery plugin he/she choose to use.
The installed proxy contains configuration informations and the protocol implementations to interact with the remote resource, thus one proxy for each shared screen to be used is needed.
Therefore the ImageViewer application can run in two modality: standalone, if there's no SharedScreen proxy installed on the ubinode; or distributed (figure 4.43).

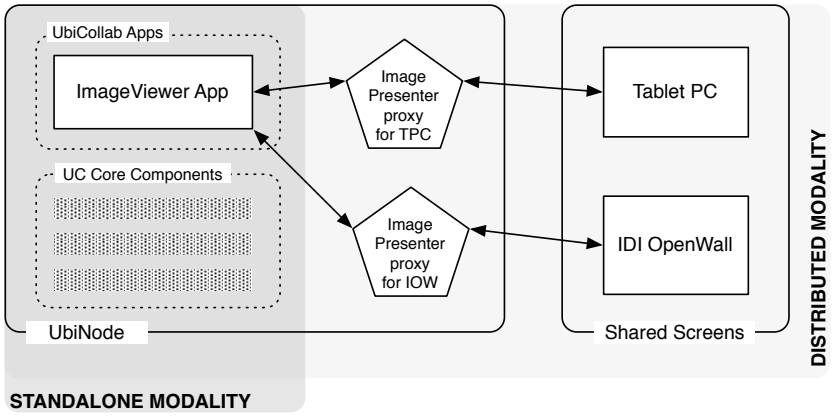



**Figure 4.43:** Standalone and distributed modality for the ImageViewer App

Our application uses proxy via the OSGi services interface. Proxies lookup is performed when the application starts or when is reactivated during the same session[34]. Class diagram for this application is reported in figure 4.44

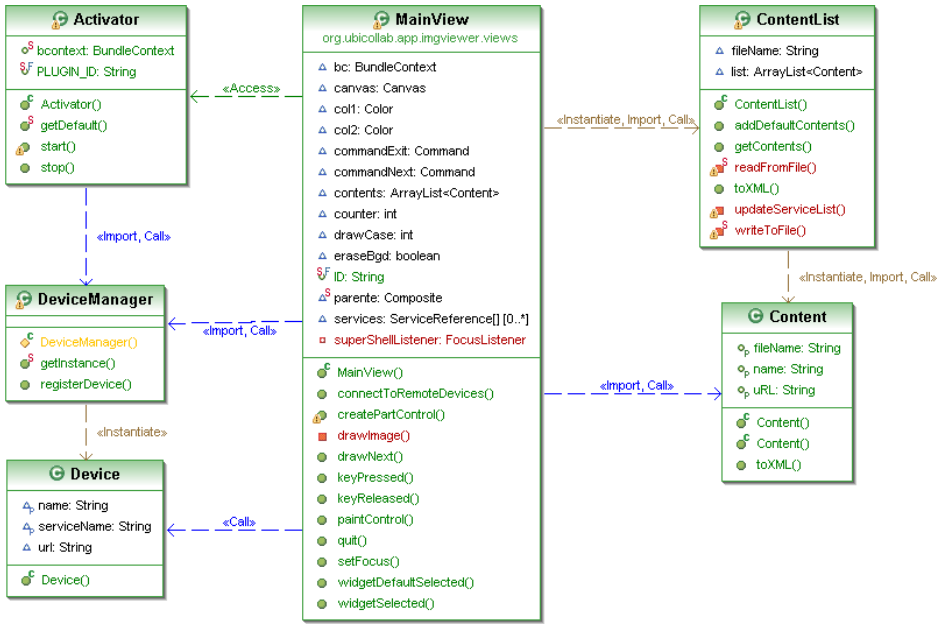



**Figure 4.44:** Class Diagram for the ImageViewer application

Images description and location are also stored in a xml file in the user space. The number of SharedScreen devices connected to the ImageViewer application is only limited by computation power of the UbiNode.

### 4.3.9 SharedScreen Proxies

The SharedScreen proxy are modules that allow the use of generic SharedScreen WebService in UbiCollab domain, exposing an interface that can be used by other modules to have access to the services provided by the remote resource via SOAP invocations (figure 4.45).

[34]Since inside the eWorkbench applications run in multitask mode they can be suspended, for discovering operations for instance, being then reactivated at the same state.

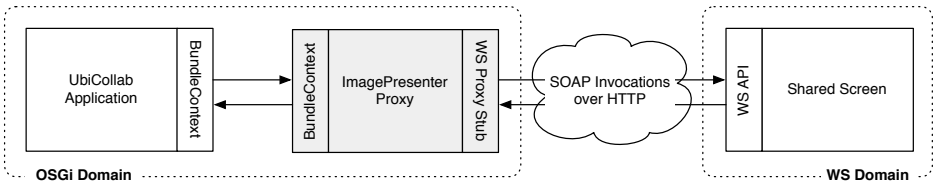



**Figure 4.45:** Communication System through proxy

The proxy routes the OSGi internal methods invocations outside the UbiNode over the network. On one side it implements an interface and register itself as service in the OSGi registry, making remote procedures accessible to other bundles; on another side it instantiate a stub (a WS Proxy) in order to communicate with external WebServices (running on shared resources) via the SOAP protocol.

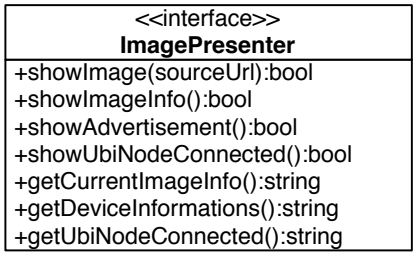



**Figure 4.46:** Interface published by the proxy

The proxy contains all the information needed to handle the connection over the network, such as the resource IP address, the implementation of the communication protocol, and routines to handle network communications errors; therefore we need to have a configured version of the SharedScreen P. for each screen we want to use.

A proxy is an independent component, once that is installed by the RDM and registered with the SDM is ready to be used as soon as an application will retrieve its *service reference*[35], an object needed by bundles to invoke methods published by other bundles. After having redeemed the *service*

[35]See the Javadoc for more informations:
http://www.osgi.org/javadoc/r4v41/org/osgi/framework/ServiceReference.html

*reference* from the OSGi service registry the Application is finally able to invoke proxy methods instantiating the interface reported in figure 4.46. The complete procedure for interfaces registration and procedure invocations is disclosed in figure 4.47.

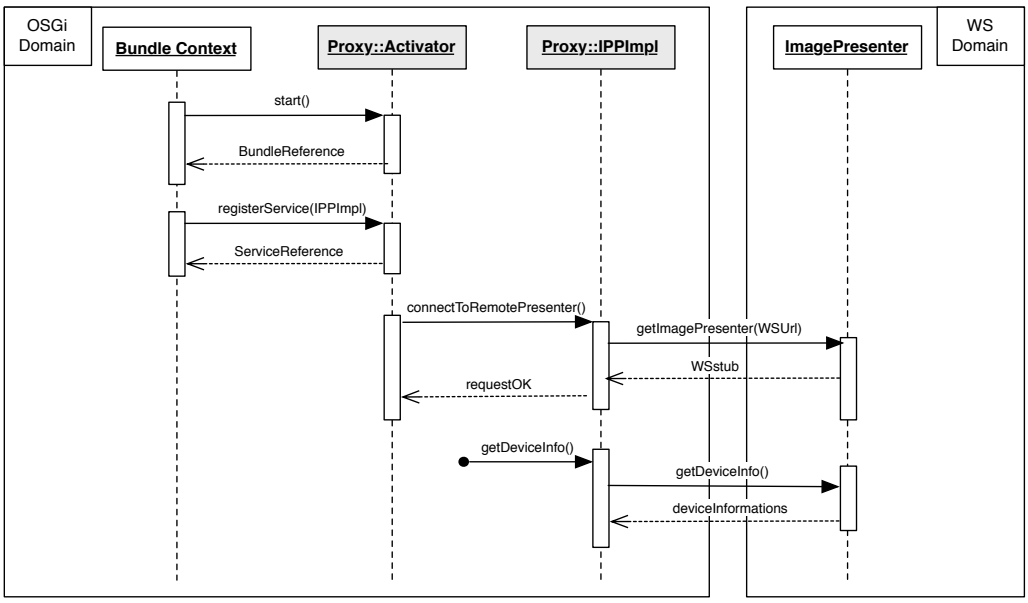



**Figure 4.47:** Sequence diagram for proxy initialization and method invocation

As support to the platform evaluation two SharedScreen proxies have been released targeting two differed type of shared screens: one for a Tablet PCs and one for the IDI Open Wall. Devices specifications and configurations are described in Appendix C.

### 4.3.10 SharedScreen WebService

In collaborative environments can be useful to have shared visual output devices such as LCD-projectors or tablet PCs. In order to use them in UbiCollab, which is a SOA-based architecture, we need have a WebService running on them listening for remote procedure calls via SOAP protocol.
The WebService implementation can be in any language because we see it

like a black box that implement a public interface declared in a WSDL file (figure 4.48), which is the only information UbiCollab needs to build a proxy for it. We assume that the WS is released by the resource manufacturer and that UbiCollab's crew has just to develop the proxy from the specifications enveloped in the WSDL file; however we developed WebServices for evaluation purposes, in order to drive shared screens hardware and test the developed proxies.

The two resources that we turned in a shared screen WebService are a Tablet PC and the IDI Open Wall; consequently two SharedScreen proxies configured for these WS have been released too. Devices specifications and used configurations are described in Appendix C.

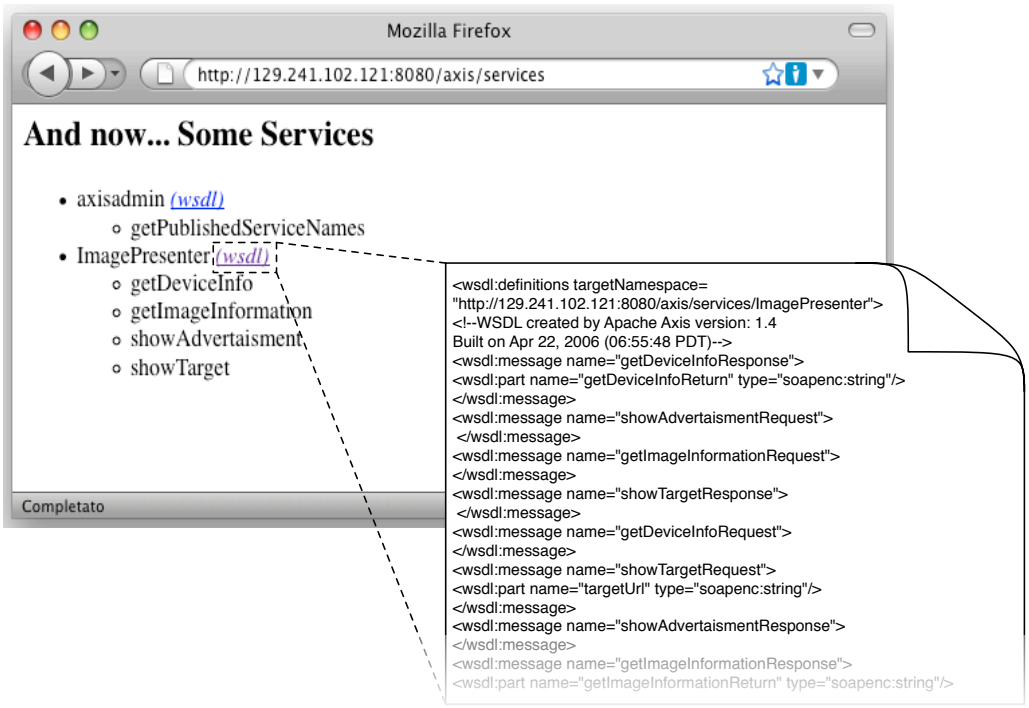



**Figure 4.48:** Service Advertisement and wsdl file for the SharedScreen WebService

Being free to decide which technology has to been employed in the WS implementation we chose to exploit our knowledge in Java technologies already used for the rest of the platform, thus we implemented WS using OSGi and

eRCP. In this environment WebService capability is provided to OSGi by the Knopflerfish Axis bundle[36] here utilized to turn our OSGi-Services in Web-Services and publish the respective WSDL advertisement file. SharedScreen service can be deployed on any device running Windows and OSGi.

Methods provided by this interface allow a proxy on a UbiNode to:

- getDeviceInfo(): get device information, such as position, ip address, owner etc..
- showTarget(): it let the shared screen show the image (.jpeg or .png extension) URL passed as method parameter.
- showAdvertisement(): this method show a resources advertisement reporting the supported gestures available for its discovery
- getImageInformation(): it return the name of the image currently presented by the device screen

Because we have developed Web-Service for two resources which serve the same purpose (show a visual output) but which deeply differs in output hardware (one has a 800x600px LCD Screen, the other has a 80x30px 201" LED screen) we had to make a proprietary implementation of the WebService Interface for each device. Class diagrams for both WS are reported in figure 4.49 and 4.50.

Tablet PC WS implementation make use of eSWT widgets and is quite similar to the implementation of the ImageViewer app.

IDI-OpenWall WebService uses the APIs available as part of the sart project[37].

[36] See the Appendix B for more informations about third-party technologies used in Ubicollab and their configurations

[37] sart project wiki: http://mediawiki.idi.ntnu.no/wiki/sart/index.php/Main_Page

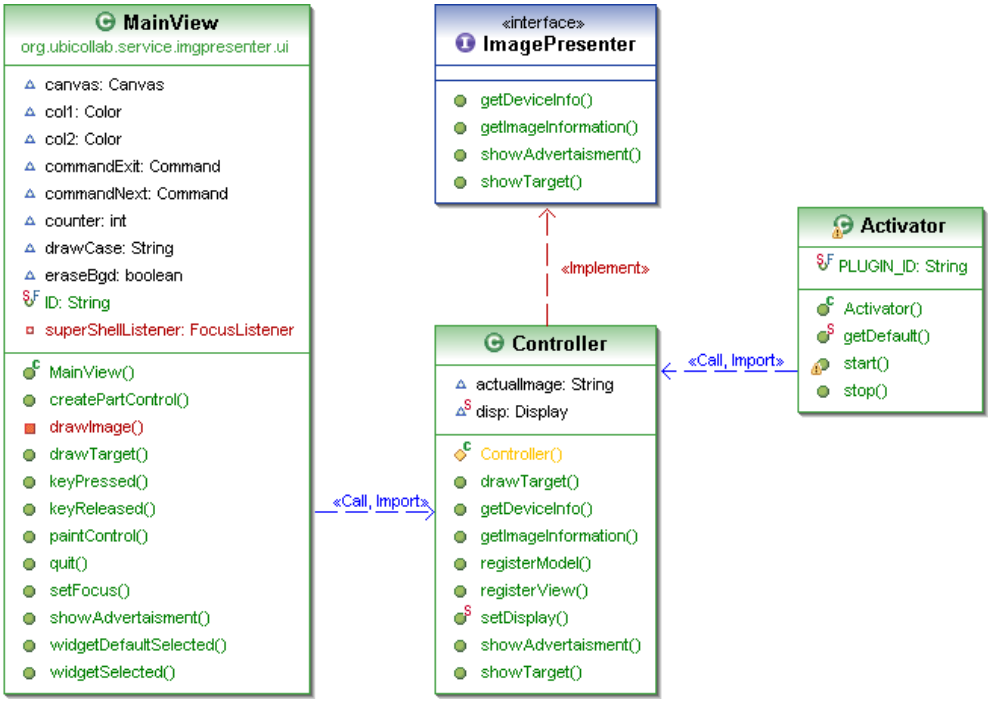



**Figure 4.49:** Class diagram, SharedScreen for Tablet PC

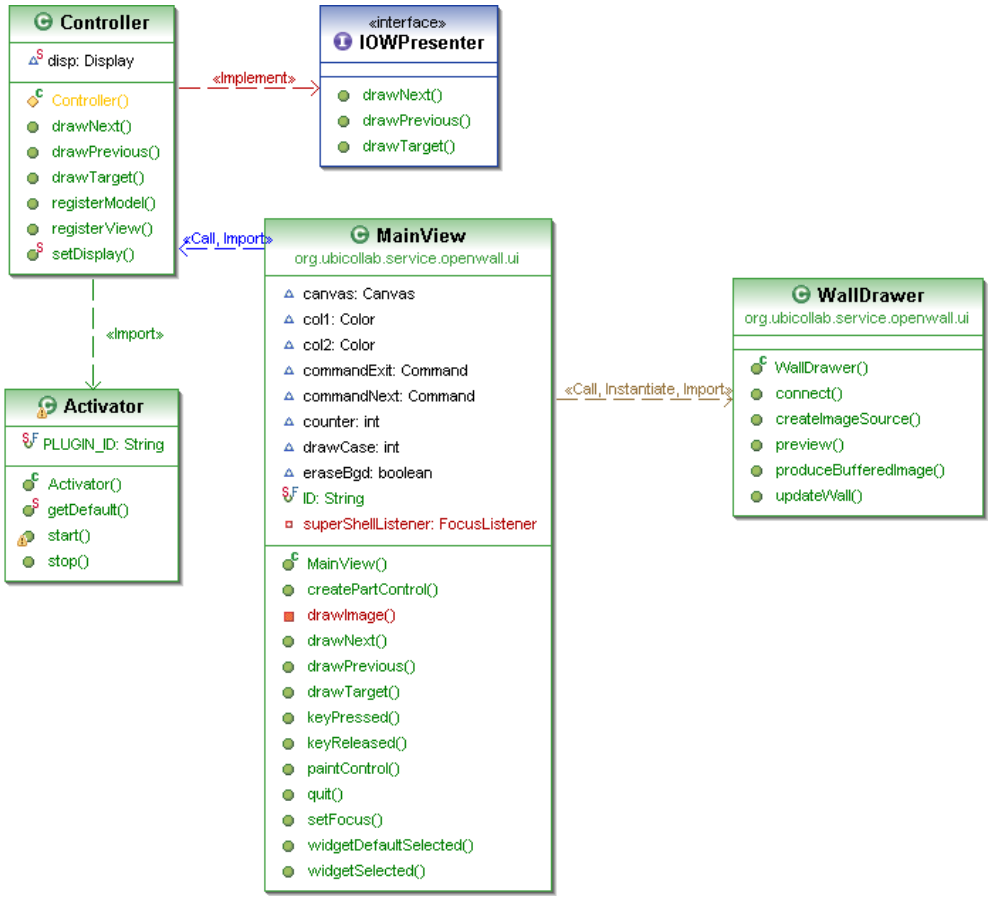



**Figure 4.50:** Class diagram, SharedScreen for IDI OpenWall

# Chapter 5

# Evaluation

In this chapter work done in this project is evaluated. The solution implemented has gone through three different evaluations: a focus group evaluation, a technical evaluation and a user testing.
The focus group evaluations has been performed in a workshop and driven by a scenario, showing a first prototype of the solution proposal. The purpose of the focus group was to evaluate the basic functionalities of the platform and decide on which aspects focus the development.
The technical evaluation consists in JUnit tests and benchmarks of the platform components which have been run during the development process.
Finally some user testing sessions have been made on a release candidate of the project.

In section 5.1 is reported the focus group evaluation.

In section 5.2 is reported the technical evaluation.

In section 5.3 is reported the user testing.

In section 5.4 requirement accomplishment will be evaluated.

## 5.1 Group Evaluation

The implemented solution has been demonstrated in a workshop the 28th of April 2009 at IDI/NTNU. People attending the event were professors Farshchian and Divitini, people for ASTRA project and a group of visiting students and researchers from University of Bergamo. As part of this demonstration a presentation was held to give an overview of the solution, before the different components were demonstrated separately. Finally, a demonstration scenario was walked through and feedbacks collected are reported.

### 5.1.1 Demonstration Scenario

*Bjorn is invited to held a presentation in a foreign university Meeting Room equipped with shared screens. The available shared screens are advertised by a four-digits code and a 2D Barcode.*
*Bjorn has never been in that room and doesn't know the specification of those displays, but he would to use them to show slides and pictures to the audience.*
*At home he uploaded slides and photos on his web personal space and he left home just with his UbiNode smartphone.*

*On the presentation day he reviews the speech in his hotel room looking to the slides showed on the smartphone with UbiCollab Image Viewer application; then he reaches the presentation room and finds two UC shared screen of different sizes. One is a tablet PC and one is a huge LED screen.*
*He decides to use them to share the slides with his audience using the smaller screen to show pictures and the bigger one to show a text related with the content.*

*He takes out from the pocket his UbiNode and runs the UbiCollab platforms*

*interacting with the system via a touch interface. He finds on both screens an advertising label reporting the two Discovering Gestures available for those devices: a numeric four-digits code and a 2D Barcode.*
*He chooses to discover both screens by typing a code because his smartphone doesn't have a camera.*
*The UC Resource Discovery Manager installs the UC proxies for those devices, establishes a communication with them and notify the user about the outcome of the operation.*

*Finally Bjorn runs the UC Image Viewer application he used earlier to review the slides, now the application recognizes and notifies the user that two new shared screen services are now available and start to communicate with them. The two shared screens get activated and start to display the contents of the presentations in a proper way. Bjorn controls and browse the slide to display interacting with his smartphone and the contents showed on the shared screens are automatically updated.*

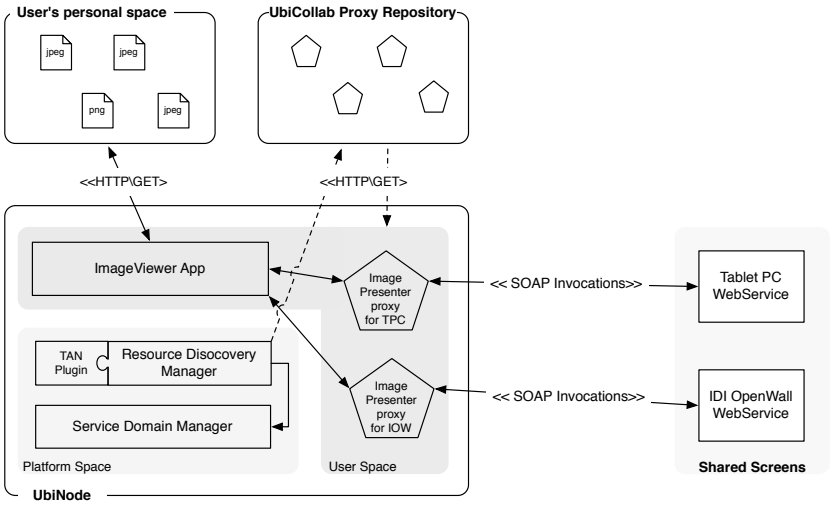



**Figure 5.1:** Demostration Scenario

### 5.1.2 Scenario Walkthrough

There are some prerequisite that have to be in place in order to play out this scenario. These prerequisite are:

- The XML file containing the service list has to be filled with proxies URIs and be uploaded on the UbiCollab server.
- Sample photos and texts have to be prepared and uploaded on my NTNU personal space used as user personal space
- The full UbiCollab platform has to be installed on the smartphone used as UbiNode
- SharedScreen WebServices has to be installed and started on both screens

In an application of the scenario to the real world the first step has to be made by the UbiCollab crew. The name "Bjorn" from the scenario is used in the walkthrough when end-user actions are performed.

1) The scenario walkthrough starts off with Bjorn launching the UbiCollab distribution on his UbiNode from his hotel room. A screenshot is displayed in figure 5.2.

2) Then Bjorn tap with his finger on the UbiNode screen and the ImageViewer app starts. ImageViewer fetch the XML file containing the location of Bjorn's remote space where the digital contents are stored, then search for SharedScreen proxy installed in Bjorn's UbiNode. Because there aren't any available SharedScreen proxies the application starts in standalone mode, as shown in figure 5.3.

3)Bjorn browse the photo taken at Trondheim, which his presentation is about, tapping once to proceed to the next photo or tapping twice to go back to the previous photo (figure 5.4). Then Bjorn packs is UbiNode and goes to the presentation place.

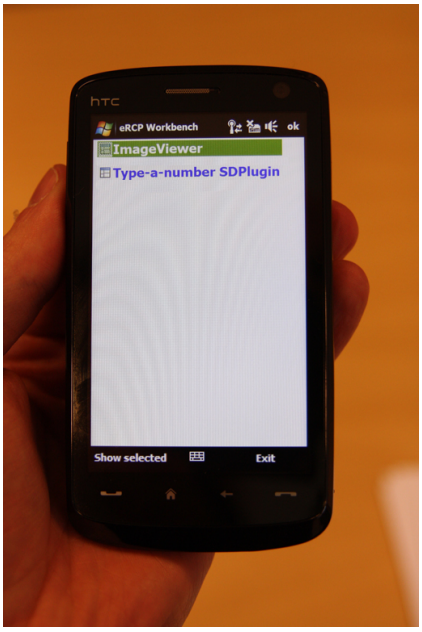



**Figure 5.2:** UbiCollab Platform Launched

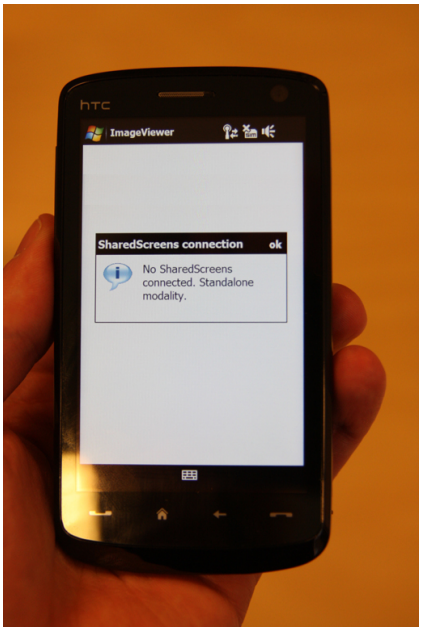



**Figure 5.3:** ImageViewer in standalone mode

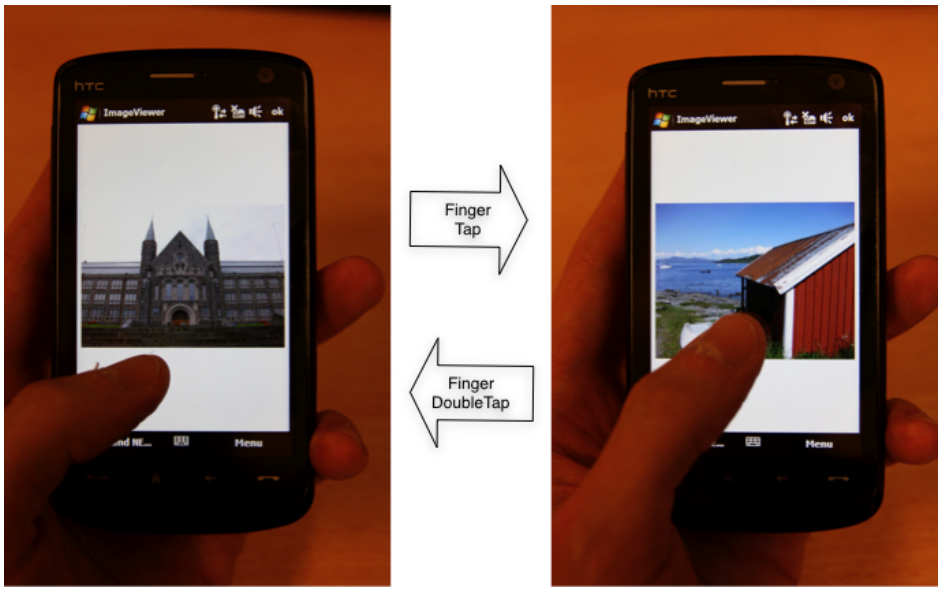



**Figure 5.4:** Photo Browsing

4)At the conference room Bjorn identify the SharedScreens by their service advertisements. Service advertisement for the tablet PC is directly in view on device screen (figure 5.5), service advertisement for the LED screen, also called IDI OpenWall[1], is revealed on a broadsheet close to the wall (figure 5.6).

5) Bjorn takes out his UbiNode and run the UC platform. At this time he choose to start using the SharedScreen tablet PC with his ubinode, he taps to select the service discovery plugin associated with the chosen discovery gesture: the Type-a-number SD Plugin. The plugin starts, fetches the service list from the UbiCollab server and show to Bjorn the GUI reported in figure 5.7.

6) Bjorn types the tablet PC advertisement code with the aid of the on-screen buttons. The system notify feedback sound for any number typed, then Bjorn push the button "DISCOVER". The plugin search in the service

[1]More informations about the OpenWall are available on the project wiki: http://mediawiki.idi.ntnu.no/wiki/sart/index.php/Main_Page


**Figure 5.5:** Resource Advertisement for the Tablet PC

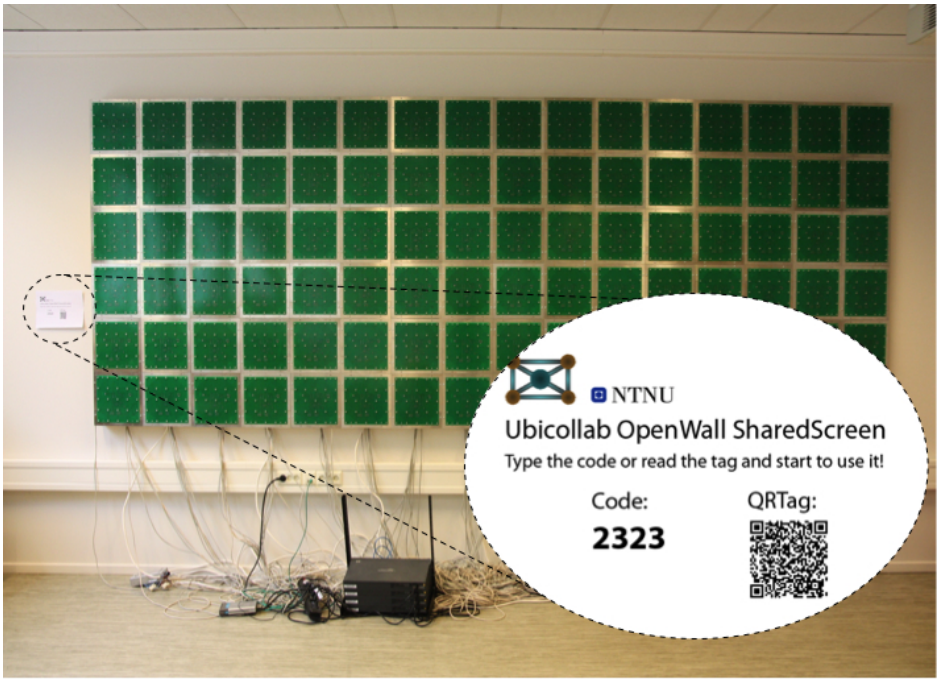



**Figure 5.6:** Resource Advertisement for the OpenWall

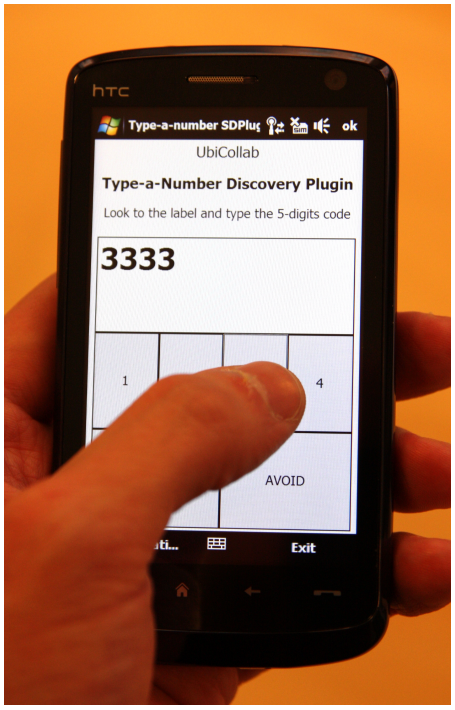



**Figure 5.7:** Type-a-number Plugin GUI

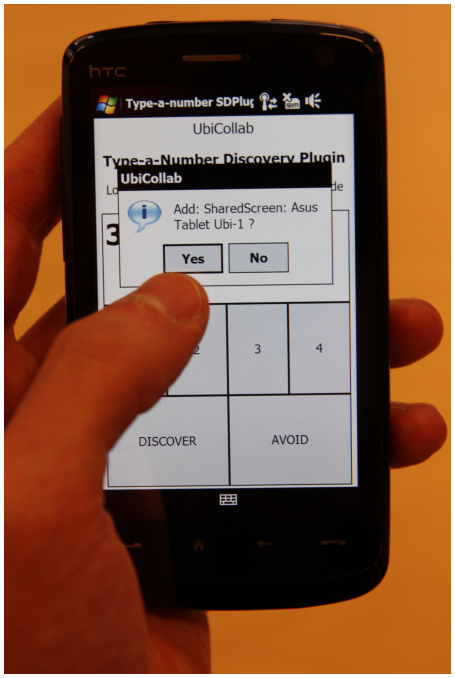



**Figure 5.8:** Resource Found Screenshot

list a resource that match the code in input and prompt to Bjorn a confirmation popup message with the name of the resource which match that advertisement code (figure 5.8).

7) Bjorn confirm the operation, the system download the SharedScreen proxy from the UbiCollab server, install and activate it; after that it shows to Bjorn a message about the successful outcome of the operations (figure 5.9).

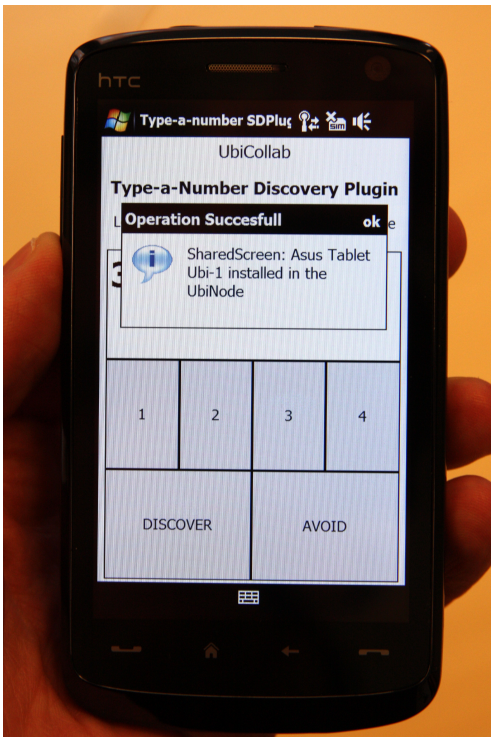



**Figure 5.9:** Resource Proxy Successfully Installed in the UbiNode

8) Bjorn goes back to the platform home window and taps on the ImageViewer App. The application recognize that a SharedScreen proxy is now available and turns on in the distributed modality showing a popup message with the name of the SharedScreen available to be used. As soon as Bjorn tap on the popup box confirming his choice the ImageViewer app show the first image of Bjorn's presentation and at the same time the tablet PC SharedScreen changes from its Advertisement mode to the active mode showing an high-resolution version of the photo which Bjorn is watching on

the UbiNode (figure 5.10).

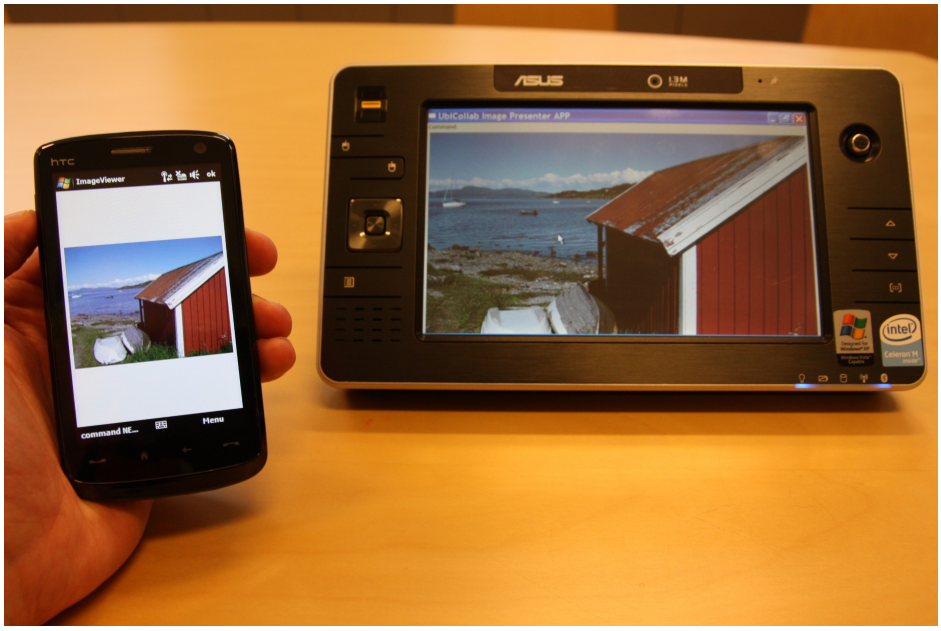



**Figure 5.10:** UbiNode and tabletPC SharedScreen showing a photo

9) Bjorn wants to use the OpenWall LED screen to show to the audience a descriptive text related to the photo reproduced on the tablet, therefore he moves to the resource discovery perspective and repeats the steps 5-6-7 keying in the OpenWall advertisement code. Finally turn on again the ImageViewer app which now show that two shared screen are available (figure 5.11).

10)Finally Bjorn is ready to show to the audience a rich version of his presentation with photos showed on the tablet PC and a related text printed on the LED screen (figure 5.12).

### 5.1.3 Feedbacks

During this demonstration the components worked as expected, without any problems. The feedback received after the demonstration was generally positives. If any remarks has not been made about the platform architecture, some hints came for the user interaction area. Professor Divitini, looking to

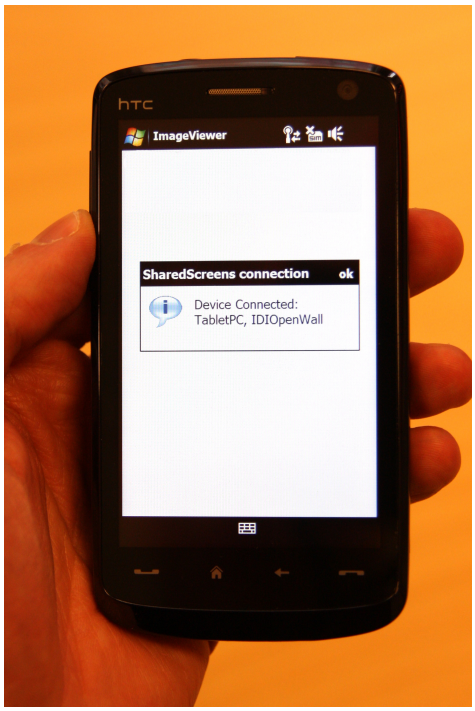



**Figure 5.11:** ImageViewer started with two SharedScreens available


**Figure 5.12:** ImageViewer with TabletPC and OpenWall Proxies Connected

implemented GUIs and mockups from the designed components, pointed out that a person without any knowledge in UbiCollab terminology cannot easily understand the meaning of proxies, resources and discovery reported in our mockups; therefore a work in achieving a more user friendly way and different abstractions to communicate these concepts has to be investigated. People from ASTRA project stated that an integration of our Service Discovery with their platform would be really interesting in order to use Discovery Gestures to improve the effectiveness of the UPnP discovery protocol currently employed in ASTRA. The possibility of creating a gateway between ASTRA and UbiCollab resource is highlighted as really interesting for both projects. Researchers from Bergamo observed that we should exploit more the environment where the user operate in order to make the resources "User Aware" and responds to him/her as soon as him/her will come close to the resource. This improvement could make the overall architecture more customized on user's habits.

As results of these feedbacks the mockups have been changed providing less but more significant informations and trying to explain with adjectives or actions terms like "Get your location proxy" instead of "GPS Proxy". A discussion about Service Discovery in ASTRA and exchange of technical concepts have been established with an ASTRA researcher. The feasibility of use WiFi network to make resources and users location awareness is started and is still under research [25, 26, 27, 28, 29, 30]

## 5.2 Technical Evaluation

### 5.2.1 JUnit Testing

Since UbiCollab is based on OSGi and thus on modules, a common JUnit testing plan is not a sufficient proof of reliability of the system, since it just can tests methods inside a module ignoring what happens outside the mod-

```
public class ResourceDiscoveryManagerTest extends TestCase {

   public void runTest() {

      ServiceReference sr_reference;
      sr_reference = Activator.bc.getServiceReference(ServiceDiscovery.class.getName());
      ServiceDiscovery rdm = (ServiceDiscovery) Activator.bc.getService(sr_reference);

      assertNotNull("A Service Reference for the bundle must be retrieved", sr_reference);
      assertNotNull("A Service Inteface for the bundle must be retrieved", rdm);

      int id = rdm.registerClient();
      boolean isInteger;
      if(Integer.valueOf(id) instanceof Integer)
            isInteger=true;
      else
            isInteger=false;
      assertTrue("The RDM correctly assign and id for the Client", isInteger);
      assertTrue("The RDM correctly register a client request",
      rdm.registerClientRequest(id, "dummy client", "dummy type", "dummy location", "dummy owner",
"dummy description"));
      assertEquals("The TestClass correctly retrieve the list of services","No matching services",
rdm.getServices(id));
      assertEquals("The RDM correctly register the RFID discovery plugin",id, rdm.registerSDPlugin
("RFID"));
      assertTrue("The RDM correcly register a service",
      rdm.registerService(id, 1, "descriptionURI", "serviceURI", "friendlyName"," type",
"description"," owner"," location"));
            }
}
```

A

B

**Figure 5.13:** JUnit Test for the Resource Discovery Manager module

ule: if the module is correctly activated in the framework and is effectively accessible by other modules that have dependencies with it.

For this reason the JUnit library has been deployed in a OSGi bundle too, therefore acting as a module that exposes a testing service to other UbiCollab modules is also able to see what happens among the modules and not just inside the modules. For instance is possible, as done for this test, to check if the service interface exposed to a module is correctly retrieved from the OSGi framework and produces the expected results.

An example of this testing approach is reported in figure 5.13

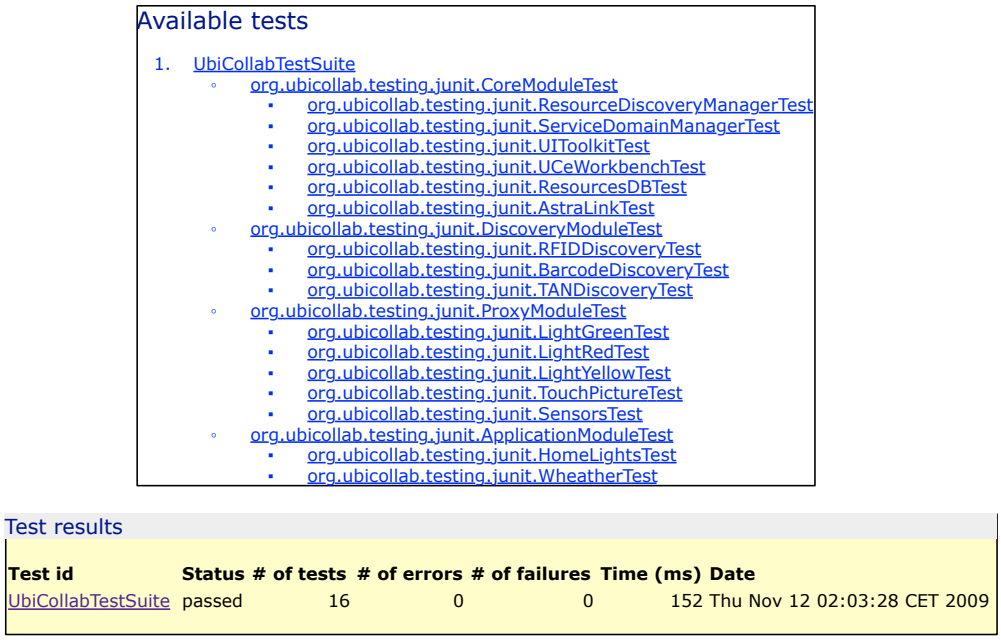



**Figure 5.14:** Output from the JUnit module

- The section A of the code tests that the RDM is correctly activated in the framework and that the service interface,which the other bundles need to consume, is correctly exposed. *Here we test what happens between the module and the rest of the framework.*
- The section B of the code is tested that the RDM interface is responding as expected to some service invocations. *Here we test what happens inside the module.*

The JUnit module has been run in the framework producing the output results reported in figure5.14

All the tests, grouped in *Test suites* for each component category are encapsulated in the module `org.ubicollab.testing.junit`

### 5.2.2 Platform Benchmark

We have tested and benchmarked our solution proposal in order to have formal evaluation of platform efficiency and provide a starting point and a comparison value for future releases.
The testbed used in our benchmark is the HTC Touch HD smartphone whose technical specification are reported in Appendix C.
We performed set of tests based on two different implementation stacks reported in figure 5.15. At the ground of the *stack A* there's the well known IBM J9 JVM, this represent the actual technology platform where UbiCollab is running on. The *stack B* probably represent the configuration where UbiCollab is moving in the close future. It is based on the new Sun's phoneME JVM that, when will have been tested for enough time without significant flaw, will substitute the J9 for its open license and better specification support. We decided to not test the platform even on different OSGi implementations since Equinox ensures the best compatibility with the eRCP framework and at the same time, according with international researches [31], it has even remarkable performances. However a previous version of UbiCollab was running well on Knopflerfish OSGi implementation, thus we can even use that configuration as a backup solution in case we won't manage to run Equinox on some hardware configurations or JVM implementations.

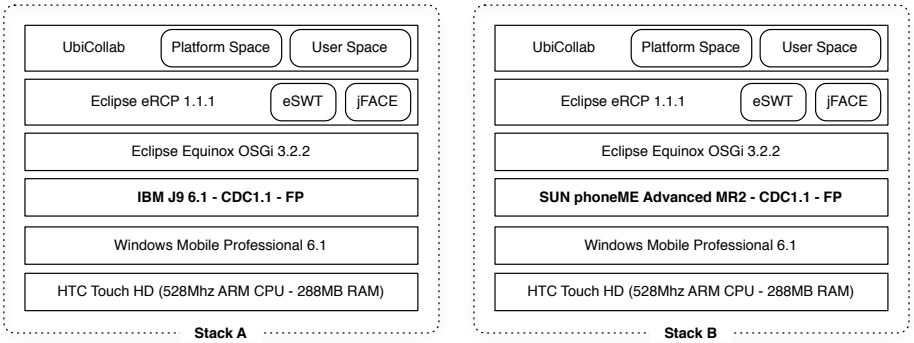



**Figure 5.15:** Implementation Stacks Benchmarked

For this test the UbiCollab stack tier has been composed in a common configuration which includes mandatory core components, one discovery plugin and one proxy service (figure 5.16).

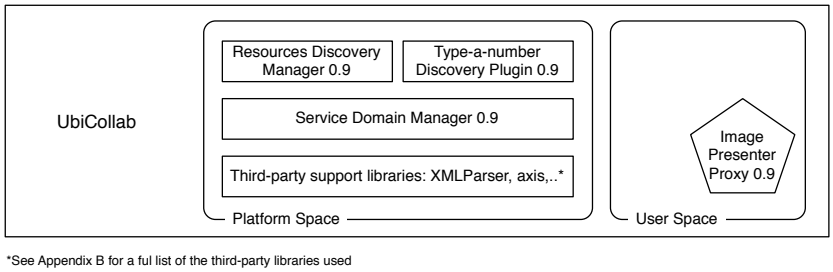



**Figure 5.16:** UbiCollab Configuration Used in the Benchmark

### 5.2.3 What we tested

We made five different benchmarks:

(a) ***Platform Space Bundles time to load and start:*** we reported and compared time taken by the mandatory core bundles to load and start on a typical platform launch. We clocked the overall time taken by platform to start, as well as time taken by each bundle[2], in order to find possible bottlenecks and check the efficiency of the Bundle startup schedule.

(b) ***User Space Bundles time to download, install, load and start:*** since functionality implemented in bundles those resides in user space are intended to be dynamically discovered and downloaded[3] by the user and not shipped as part of the standard distribution, tests of these bundles include also time to download and install them inside the framework.

[2]Time taken by third-party bundles to load and start include an overhead time due to the initial OSGi and eRCP framework startup. This overhead doesn't significantly longer affects UbiCollab bundles since they have a lower startup priority (they're loaded after mandatory third-party bundles).

[3]Via HTTP protocol

Furthermore we even keep track of data transmitted and received over the network. This marker has also to be taken in account since wireless network where UbiNodes operate such as WiFi or UMTS/GPRS[4] ones could have a limited bandwidth as well as pay-as-you-use data plans.

(c) ***CPU rate of utilization:*** This is an important marker for the reason that is highly related with device autonomy. According with our HTC's specifications, our devices allow a stand-by time up to 440hours. Since processor load in stand-by is between 1% to 5%, and reminding that CPU usage and autonomy are competing issues in mobile environments, our goal would be to optimize our architecture to achieve the lowest CPU usage for the same service level and thus expand device's autonomy when is unplugged from the power-grid.

(d) ***Physical Memory Usage:*** This marker has an evident impact since keeping the complete platform compact is a remarkable goal for every mobile applications. Because UC and third-party technology implementations are the same for both stacks, this marker totally lever on the Java Virtual Machine implementation.

(e) ***Central Memory Usage:*** This index is relevant for devices compatibility and reliability points. First off compatibility. Our testbed come with huge amount of RAM for a smartphone, but Sun's specification allow CDC JVMs running on devices equipped with a much smaller amount of memory: 2Mb[5]. Allowing UbiCollab running in a tight quota of RAM permit us deployment on a wider number of devices including older ones. Hence, monitoring memory usage, we can notify potential system flaws since an abnormal memory usage characterized by an high number of usage peaks and consequent resource saturation besides being an indicator of bugs or poor code optimization can tamper with background process

[4]Universal Mobile Telecommunications System/General Packet Radio Service are two popular transmission protocols allowed by cellular provider to exchange data over 3G/GSM networks

[5]See chapter 4 for more technical details

like OS system notifications crashing or making unstable the device.

### 5.2.4 How we tested

Bundle operations inside the OSGi framework have been clocked running Equinox in debug mode, recording and comparing system's timestamps. CPU and memory usage data have been logged (using a two-second resolution) by the software acbTaskMan pro 1.4.1[6] running in background inside the Windows Mobile stack tier.

[6]acbTaskMan, Acbpocketsoft - www.acbpocketsoft.com

### 5.2.5 Results and conclusions

Tests have been performed interacting with the UbiNode for 72 seconds, making common operation, like discovering resources and managing services.

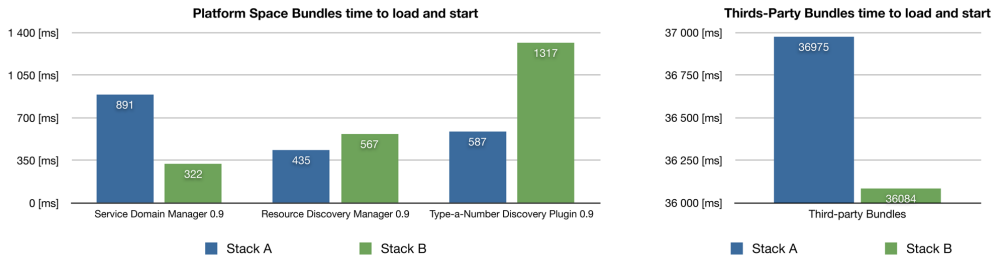



**Figure 5.17:** Benchmark's Results

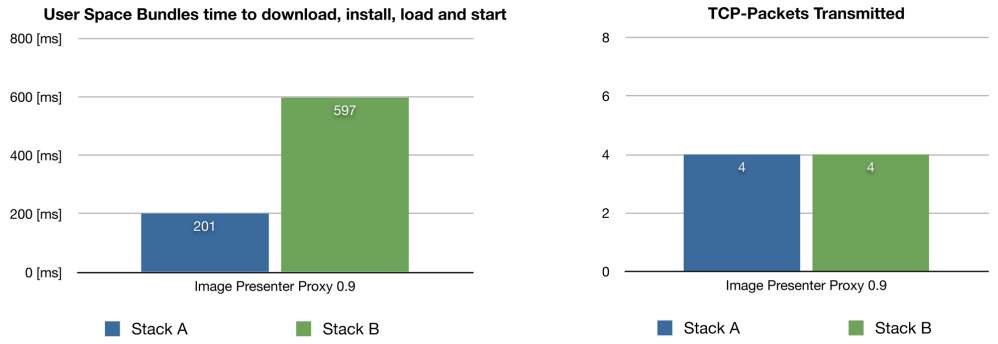



**Figure 5.18:** Benchmark's Results (2)

Results confirm the overall good performances of both stacks.
J9 JVM at the ground of stack A is faster in computing discovery operations (test b): this result can be justified since those are really resources high-consuming operations which involve third-party technologies for making SOAP invocations and connect to networks interfaces; the more experienced J9 probably can better serve requests and notification among different components. The new phoneME at the ground of stack B uses 1/3 of physical space occupied by J9, takes 1 second less than J9 to complete the platform launch and generally let the platform feel more reactive to the user.

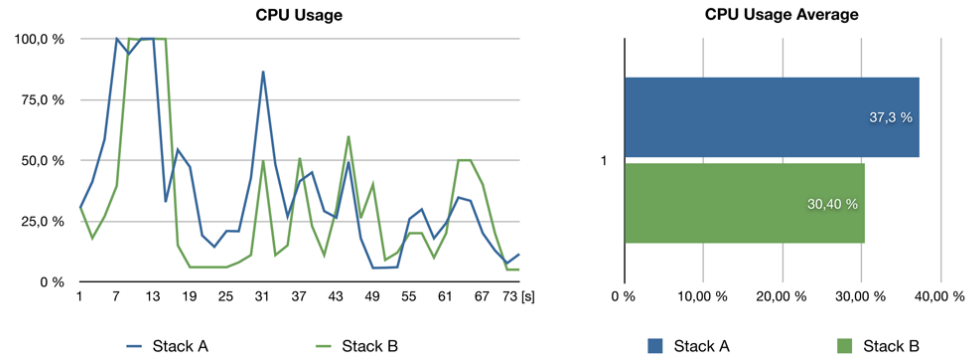



**Figure 5.19:** Benchmark's Results (3)

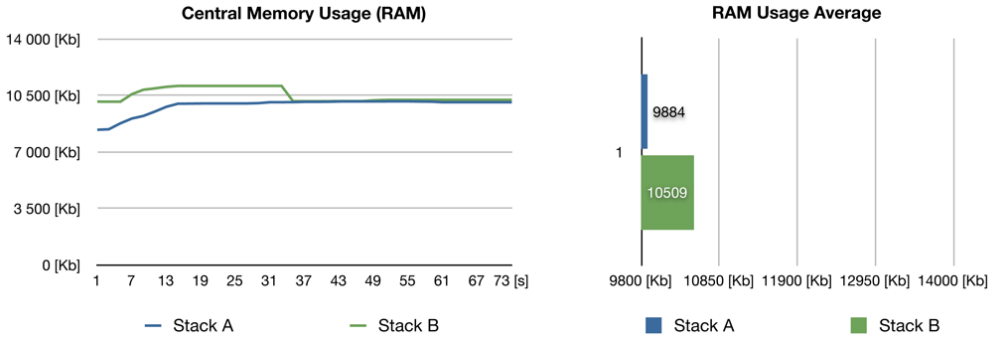



**Figure 5.20:** Benchmark's Results (4)

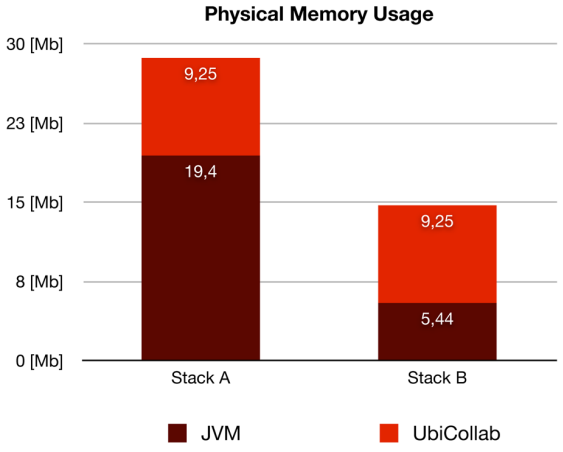



**Figure 5.21:** Benchmark's Results (5)

With some optimizations to fix the discovery bottleneck highlighted in test b can be ready to became the official Virtual Machine used for the UbiCollab platform.

## 5.3 User Testing

User testing have been made in order to proof system's reliability and collect feedbacks and suggestion in the usability area.
The tests took place in November, 2009 at IDI-NTNU[7] and at SINTEF[8]

After a brief introduction about UbiCollab, people attending the test were asked for running the system and complete some discovery operations which involves the use of resource discovery plugins, applications and proxies. All the operations were driven by the scenario reported chapter 1. The test involved the use of some prototypical home-automation and health-care devices.

We recall to the reader the scenario reported at pag. 3 presenting it schematically in figure 5.22 and 5.23

Applications and proxies developed for the user testing like the *HomeLights app* and *LifeShirt app* are still under testing and will be soon released.

Each test session lasted roughly 30-40 minutes, after that the user was asked to compile a questionnaire evaluating his/her understanding of the system and the level of user-friendliness experienced with it.

Feedbacks received were quite good, both in usefulness and usability areas. Despite a slow response of the GUIs to some user actions the system has shown an overall good reliability.

[7]Department of Information and Computer Science, Norwegian University of Science and Technology (http://idi.ntnu.no)

[8]The SINTEF Group is the largest independent research organization in Scandinavia. (www.sintef.no)

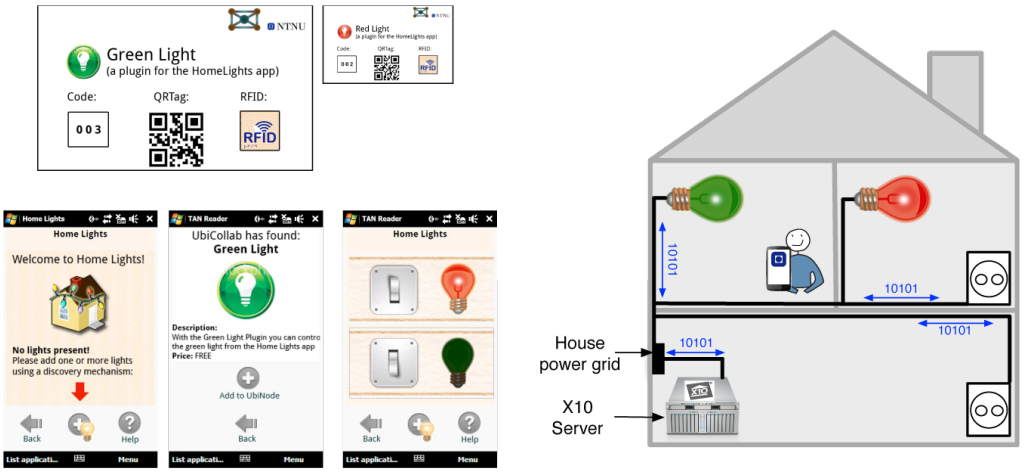



**Figure 5.22:** First part of the Scenario,home-automation devices interaction

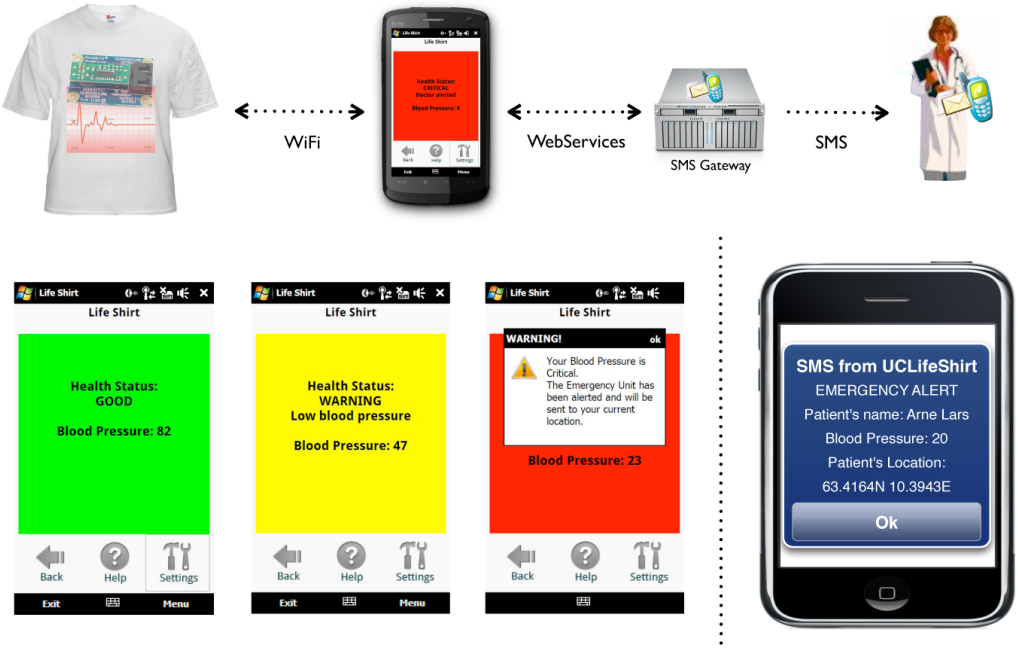



**Figure 5.23:** Second part of the Scenario, remote personal health-care devices interaction

Opinions gathered from the questionnaire will be analyzed and debated in the following months in order to review the work done and find ideas for future platform improvements.

In the following pages the reader can find some questionnaires randomly chosen from the ones collected.

# UbiCollab Evaluation Questionnaire

For evaluating resource discovery and composition in UbiCollab
Date: Nov. 09, 2009
Location: SINTEF ICT
Please return to Babak Farshchian.
Please respond to the following questions.
Please use the back of the sheet to write any comments you might have. Any feedback will be relevant

**Table 1**

| Question | 1 (strongly disagree) | 2 (Disagree) | 3 (No opinion) | 4 (Agree) | 5( Strongly agree) |
|---|---|---|---|---|---|
| **Understanding** | | | | | |
| It was easy to understand the concept of a UbiCollab application | | | | | X |
| It was easy to understand the concept of a discovery plugin | | | | | X |
| It was easy to understand the concept of a proxy service | | X | | | |
| It was easy to understand what I was doing | | | | | X |
| It was easy to understand why I was doing this | | | | | X |
| Overall, it was clear for me what the system did and why | | | | | |
| | | | | X | |
| **Usefulness** | | | | | |
| Using the system gives me greater control over my devices | | | | X | |
| Using the system will improve my interaction with devices | | | | | X |
| The system addresses an important need | | | | X | |
| It is easier to use this system than other systems I know of | | | X | | |
| Overall, I find the system useful for interacting with devices | | | | | X |
| | | | | | |
| **Usability** | | | | | |
| I made a lot of errors while using the system | | | X | | |
| I was confused when using the system | X | | | | |
| Interacting with the system required a lot of mental effort | X | | | | |
| It was easy to recover from errors when using the system | | | X | | |
| The system behaved in unexpected ways | X | | | | |
| The barcode reader was easy to use | | X | | | |
| The type-a-number (TAN) reader was easy to use | | | | | X |
| The RFID tag reader was easy to use | | | | | X |
| The web-based application was easy to use | | | | | |
| It was easy to control the lights using the system | | | | | X |
| It was easy to navigate among the ~~screens~~ perspectives | | | | | X |
| Overall, I found the system easy to use | | | | | X |

The system did (some)

# UbiCollab Evaluation Questionnaire

For evaluating resource discovery and composition in UbiCollab
Date: Nov. 09, 2009
Location: SINTEF ICT
Please return to Babak Farshchian.
Please respond to the following questions.
Please use the back of the sheet to write any comments you might have. Any feedback will be relevant

**Table 1**

| Question | 1 (strongly disagree) | 2 (Disagree) | 3 (No opinion) | 4 (Agree) | 5( Strongly agree) |
|---|---|---|---|---|---|
| **Understanding** | | | | | |
| It was easy to understand the concept of a UbiCollab application | | | | X | |
| It was easy to understand the concept of a discovery plugin | | | | X | |
| It was easy to understand the concept of a proxy service | | | X | | |
| It was easy to understand what I was doing | | | | X | |
| It was easy to understand why I was doing this | | | | X | |
| Overall, it was clear for me what the system did and why | | | | X | |
| | | | | | |
| **Usefulness** | | | | | |
| Using the system gives me greater control over my devices | | | | X | |
| Using the system will improve my interaction with devices | | | | X | |
| The system addresses an important need | | | | X | |
| It is easier to use this system than other systems I know of | | | X | | |
| Overall, I find the system useful for interacting with devices | | | | X | |
| | | | | | |
| **Usability** | | | | | |
| I made a lot of errors while using the system | | | X | | |
| I was confused when using the system | | X | | | |
| Interacting with the system required a lot of mental effort | | X | | | |
| It was easy to recover from errors when using the system | | | | X | |
| The system behaved in unexpected ways | | X | | | |
| The barcode reader was easy to use | | | | X | |
| The type-a-number (TAN) reader was easy to use | | | | X | |
| The RFID tag reader was easy to use | | | | X | |
| The web-based application was easy to use | | | | X | |
| It was easy to control the lights using the system | | | | X | |
| It was easy to navigate among the screens | | | | X | |
| Overall, I found the system easy to use | | | | X | |

# UbiCollab Evaluation Questionnaire

For evaluating resource discovery and composition in UbiCollab
Date: Nov. 09, 2009
Location: SINTEF ICT
Please return to Babak Farshchian.
Please respond to the following questions.
Please use the back of the sheet to write any comments you might have. Any feedback will be relevant

**Table 1**

| Question | 1 (strongly disagree) | 2 (Disagree) | 3 (No opinion) | 4 (Agree) | 5( Strongly agree) |
|---|---|---|---|---|---|
| **Understanding** | | | | | |
| It was easy to understand the concept of a UbiCollab application | | | | | X |
| It was easy to understand the concept of a discovery ~~plugin~~ service | | | | X | |
| It was easy to understand the concept of a ~~proxy~~ service | | | X | | |
| It was easy to understand what I was doing | | | | X | |
| It was easy to understand why I was doing this | | | | X | |
| Overall, it was clear for me what the system did and why | | | | X(*) | |
| | | | | | |
| **Usefulness** | | | | | |
| Using the system gives me greater control over my devices | | | | X(**) | |
| Using the system will improve my interaction with devices | | | | X** | |
| The system addresses an important need | | | | X** | |
| It is easier to use this system than other systems I know of | | | | X | |
| Overall, I find the system useful for interacting with devices | | | | X | |
| | | | | | |
| **Usability** | | | | | |
| I made a lot of errors while using the system | | | X | | |
| I was confused when using the system | | | X(**) | | |
| Interacting with the system required a lot of mental effort | | X | | | |
| It was easy to recover from errors when using the system | | | X(**) | | |
| The system behaved in unexpected ways | | X(**) | | | |
| The barcode reader was easy to use | | | * | X | - after a short training |
| The type-a-number (TAN) reader was easy to use | | | | | X |
| The RFID tag reader was easy to use | | | | X | - after a short training |
| The web-based application was easy to use ? | | | | | |
| It was easy to control the lights using the system | | | | | X |
| It was easy to navigate among the screens ? | | | | | |
| Overall, I found the system easy to use | | | | X | |

(*) except the feedback ~~what~~ is where the service is installed
(also a short description of the service might be helpful)
(**) it depends, how will the services are implemented and what are the alternatives
(***) sometimes

(* see **)

## 5.4 Requirement Fulfillment Analysis and improvement suggestion

Analyzing the results from the evaluations, I can assert that all of the requirement listed in Section 2.3 have been partially or totally accomplished. The requirement partially satisfied and motivations are the following:

- AR-MB-04 (accessing to mobile device features): At the present time we just manage to use embedded device speaker and camera resources, it would be really interesting, for new UI mechanism development, manage to have access even to embedded accelerometers (a feature that is becoming popular on high-end smartphones) and to the device microphone.
- AR-MB-05 (code optimization): Components code has to be better optimized for mobile, the startup launch time of the platform is still too high compared on an average typical Windows Mobile application. Code should be written using procedure less resources-demanding and comply with to Java 1.4. Component implemented in Java 1.6 should be rewritten in order to avoid the use of converters like retroweaver[9] and thus boosts performances.
- FR-UI-01 (user can set default UI): At the present time the User cannot choose a default interaction mechanism, this is a feature that can be easily implemented in the future.
- AR-DE-04 (UIs user friendliness): Even if a lot of steps in usability have been made, from the prototype used in the focus group to the system released for the user testing, GUIs and others interaction mechanisms still need to be improved to better match users' behaviors.
- FR-RD-06 (SDPlugin can be mutually discovered): In the current release Service Discovery plugin can discover only proxies, since they are

[9] See Appendi C: Ubicollab distribution and versioning for more details

stored in the user space. Because RDPlugins are intended to reside in the Platform Space a mechanism to allow this discovery operation and separate it from the proxy discovery one.

# Chapter 6

# Conclusion and future research

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

In section 6.1 the contribution this project has provided to UbiCollab will be described.

In section 6.2 problem encountered during the work will be explained.

In section 6.2 a short evaluation from the writer point of view of work results will be given

In section 6.3 some suggestion for future research project and technical improvement within UbiCollab project will be presented.

## 6.1 Contributions

The work started with readings about UbiCollab existing architecture [1] and third-party technologies used in: Java, OSGi, Web Services; then my focus moved on topics assigned me for the diploma task:

The contribution that my work has provided to UbiCollab is here listed:

*Research on User Centered Service Discovery protocols and Object Tagging:* The study about user interaction in UbiCollab started with a research about user-centered service discovery protocol and object tagging technologies; in fact before starting to interact with a resource we must recognize it and give to our system the information needed to download and install a proper proxy in order to use the resource, these informations are wrapped in a physical tag that could be a label with a number, a barcode or a RFID tag. I've evaluated this technologies and decided in which direction focus the research on.

*Research, comparison and evaluation of different Java Virtual Machines for handheld devices:* In order to read a tag and use resources that can be found on our way we need to deploy the system on mobile device, moreover the User-Centered paradigm implies that all the core components of the system belong to a unique user who has to have a full control on them. I've evaluated different solutions in order to run our java based core on smartphones that fit the requirements of usability we are pursuing.

*Research, comparison and evaluation of different User Interaction technologies:* After the elaboration of a user-centered service discovery concept I analyzed the way in which the user would interact with the system to perform a resource discovery operation, following the direct-manipulation paradigm.

*Research, comparison and evaluation of technologies for Graphical User Interfaces development:* Even if I designed an UI solution which supports multiple sort of User Interaction, the common denominator for them is the presence of a GUI used to interact with and return feedbacks. Therefore a review and a comparison of the actual tools for building GUI in java mobile environment has been produced.

*Elaboration of a test Scenario for platform evaluation purpose:* Since the UbiCollab overall discovery subsystem was not still tested on mobile devices I elaborated a scenario in order to test modules from past contributions, mod-

ules I developed and their integration. The scenario elaborated has driven the development of application and proof-of concepts modules and the architecture evaluation.

*Update of the UC modules developed in previous works and standardization of the module unit:* the change of the platform running environment: from desktop to mobile has induced a change of the OSGi container and Java Virtual Machine, therefore the existed Service Discovery modules has been updated in order to adhere the new container and new VM constrains. During the update work the module structure (in term of package and file internal distribution and naming) has also been updated, thus the a new module structure and naming conventions has been defined in order to provide a standardized module architecture to future works.

*The eWorkbench* is the central component of User Interaction solution in UbiCollab. It acts as User Interaction service provider supplying to other modules a plugin mechanism to let them publish their proprietary UIs engines as soon as they are discovered by the RD subsystem and without charging any configuration process to the user. Moreover this approach let UI can be developed without extensive coding since new UIs mechanism can wrapped in the existing modules.

*The Type-a-Number Discovery Plugin* is a plugin for the Service Discovery subsystem which allow to install a shared resource tagged with a four-digits code that can be used by application in the UbiCollab domain. It comes with a touch-based GUI designed following design patterns elaborated during the theoretical work.

*The RFID Discovery Plugin* is a plugin for the Service Discovery subsystem which allows to install a shared resource tagged with a RFID tag by the use of an embedded or external RFID reader.

*The 2DBarcode Discovery Plugin* is a plugin for the Service Discovery subsystem which allows to install a shared resource tagged with a QR/Datamatrix

Barcode taking a photo of it with the embedded smartphone camera.

*AstraConnector* is a core component for sharing and integrating services with the ASTRA Project.

In order to test the SD subsystem and the UI framework following the elaborated scenario we designed several modules:

*The ImageViewer app* is an application shaped on the idea to absolve the task to show and browse pictures stored on a remote space by an handled device. Pictures are shown on the device screen and can be browsed tapping on the screen with fingers. Moreover ImageViewer can connect and control shared screen devices in order to share user pictures to an audience. ImageViewer can drive multiple external device simultaneously and show a version of the picture adapted for the output device.

*The SharedScreen Proxies* are modules that allow the use of generic SharedScreen WebService UbiCollab domain, exposing an interface that can be used by other modules to have access to the services provided by the remote resource via SOAP invocations. In order to be used by an UC app they have to be discovered by a Service Discovery Plugin (such as the Type-a-Number on presented earlier). A proxy is configured map one-to-one interaction with the WS for which is configured for

*The SharedScreen WS for tablet PCs* is a WebService deployable on table PCs which provide a WS interface and thus methods to control the reproduction of pictures on the PC remotely. Since it is a standard WS it can be used by any software which follow the specification enclosed in its wsdl file.

*The SharedScreen WS for IDI OpenWall* is a WebService with provide the same interface of the tablet PCs edition but which differ in implementation since it has to connect to a really unique device. For other aspect is equal to the one for tablet PCs.

*Setup of the platform on mobile devices and benchmarks:* finally the component developed have been deployed on mobile devices and tested. During

the tests resources usage data from the user device have been collected and discussed.

## 6.2 Problems Encountered

Problems have been encountered mainly in the engineering part of the work. UbiCollab components works with several third party components. These software are not designed to work together and thus find a stable configuration has not been trivial. For instance the Apache Axis module we are using to provide WebServices capability to our components has been revealed incompatible with the last release of eRCP (1.2) therefore, since the WS capabilities is a mandatory requirement and we haven't found a valid alternative, we have to use the previous version of eRCP (1.1). This bug has been submitted to the Eclipse foundation and we hope will be fixed in the next eRCP release. Another engineering problem concerned the eWorkbench. This component, provided as a customizable draft, is not well documented. Hence we had to do some reverse engineering on it in order to understand methods and procedures logic and thus being able to adapt it for our platform.

## 6.3 Evaluation

The assignment for this thesis was to assists the developing of Service Discovery in UbiCollab working on the User Interaction aspects. This tasks brought me to spread my work in different areas, from design patterns for GUI layout (concepts related to Usability topics and theoretical user behavior analysis) to strictly technical issues like Virtual Machine implementation and dependency conflict resolution. Because I didn't have any previous knowledge in almost all of the concepts involved, both theoretical and technical, this work has been a big challenge for me and I learned really much from it. Moreover the scientific approach to problems patiently taught me, the evaluation of

solutions elaborated, criticism and merits received during the report work by my supervisor Babak Farshchian and co-advisor Monica Divitini will help me in my work life in any kind of project I will have to work with. For this reason I'm soundly grateful to them.

After having performed a deep evaluation of the implemented solution, the conclusion is that this work has successfully achieved its goal. A user interaction framework for UbiCollab has been designed, implemented and evaluated.

At the time the project assignment was agreed and the master contract was signed, it was clear that the amount of work associated with user interactions would be quite large, thus I had to make some choices. I chose to define a framework to support future contributors in the development of their own proprietary mechanism instead of focusing on the development of my own interaction way; therefore the main regret is that I haven't had enough time to experiment some innovative interaction mechanisms such us free-form gestures which I depicted in the problem evaluation. However this additional work will be part of my summer job and will be reported for the UbiCollab project.

At the end of my research period I can say that I've set the fundamentals for an efficient user interaction, but a lot of work has still to be done to improve the user experience with UbiCollab. It's an hard challenge, but the number of applications suitable for this scopes and the needs that can be addressed by our collaboration system worth to spend time and resources on it.

## 6.4 Future Works

In this section some ideas for future works will be suggested. These ideas have been categorized into project suggestion (elements in the service discovery field) and Technical Improvements (suggestions about third party technologies adoptions).

***Suggested Research Projects:***

**Service Discovery:** Looking to the future the use of the Discovery Gesture should be just the last ring of the discovery chain and would be used just for the proximity resource discovery, when we have the resource on sight. Indeed when we are far from the resource we're looking for and therefore this one cannot be seen, we would use a sort of virtual 2D/3D navigation-based discovery that make use of "Virtual Tags". An approach to implement these concepts would be to make services and resources location aware by a scalar technology that uses GPS, Wifi Triangulation/Fingerprinting[28], and a Discovery Gesture for dynamically estimate the mutual position of services and users, giving back an overview of services closer to the position[6, 35]

**Service Discovery Plugin:** New discovery gestures has to be invented and implemented. For their contiguity with Ubiquitous Computing concepts in terms of hiding and pervasive computation power, Free-form gestures can be a extremely interesting field of research. Sun, with its Sunspot technology[1], is providing to developers and scientist a plumb device for implementing and testing new gestures. Moreover Sunspot technology accomplish UbiCollab opensource license hence can be distributed with it.

**Proxies:** We need more proxies both for resources and services. Proxy for services can be easily implemented from WebService freely available on internet like Weather Forecast webservices, as the one provided by AccuWeather[2]; or finance services as the one provided by Yahoo[3]. Because location awareness is everyday raising in importance a popular resource proxy can be a GPS proxy

***Technical Improvements***

**Remote OSGi:** R-OSGi is a middleware which extends the centralized standard OSGi service registry to support distributed module management.

[1]Sunspot technology: http://www.sunspotworld.com/
[2]AccuWeather: http://www.accuweather.com/
[3]Yahoo Finance: http://developer.yahoo.com/finance/

It is deployed and looks like a conventional OSGi bundle, but it turns standard application into distributed applications by simply indicating where the different module should be deployed. During benchmarks [32] this new distributed approach has shown better performance than famous competitors such as RMI or UPnP. It can be employed alternatively to Axis to provide distributed computation capabilities

**Qt:** **Qt**[4] (pronounced as the word "cute") is a cross platform UI framework that is earning popularity among developers since it provide tools to build really fancy and powerful GUIs. Born for developing in C++ language with a proprietary look and feel, now a Java porting is officially supported as well as the use of native widget. If a porting for OSGi would be feasible Qt libraries could improve a lot GUIs usability.

**WS-SOAP provider:** Axis v.1.x (Axis v1) is currently used as the core engine to provide Web services in UbiCollab, but this product has been discontinued since April 2006. One of the major problems with the Axis v1 is that the "rpc/encoded" format is used in SOAP encoding instead of the "document/literal" format. "rpc/encoded" has been deprecated by the WS-I Basic Profile[5], which is the baseline for inter-operable Web services. Axis v1 is for instance not inter-operable with Axis v2 (a redesign of Axis v1, with the support for SOAP1.2/REST/and other newer standards). In addition Axis v.1.2 and older versions (like the port to OSGi which is used in UbiCollab) have problems with the changes in the XML-handling that were introduced with Java v1.5.x (JAX-RPC 1.1 specifications). Axis 1.x is even incompatible with the latest eRCP distribution, thus another solution has definitely to be investigated. Knopflerfish has developed a bundle of Axis 2.x, thus its compatibility with Equinox OSGi should be tested. An alternative to Axis 1.x can also be kSOAP[6]: another WS provided as OSGi bundle.

[4]Qt Software: http://www.qtsoftware.com/products/

[5]http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile

[6]KSOAP: http://ksoap2.sourceforge.net/

# Appendix A

# Task Assignment and Scenario

## A.1 Project description

Resource discovery has become the cornerstone of ubiquitous computing. Connected devices and objects in our physical environment necessitate mechanisms for finding these objects before being able to use them. Resource discovery (also called service discovery) is the common term used to denote technologies that assist us in finding/discovering available services/resources /objects around us. The objective of this project task is to further enhance UbiCollab Resource Discovery Manager subsystem. The focus in UbiCollab is to implement a user-friendly and user-centered resource discovery mechanism. Conventional discovery technology is focused on machine-to-machine resource discovery, allowing for various levels of self-configuration of networked services. In UbiCollab we focus on resources/services that are provided, advertised and used by end users in online communities.

**Scenarios that can be supported by such a system include:**

- Sharing of user-generated content:
    * Music files in an MP3 player are provided in form of services that

a group of friends in a party can access in order to listen to music.

  * A shop owner in the town can provide his/her product catalog in form of a service that can be discovered and browsed by potential customers.

  * A user can dynamically set up a photo sharing service on her mobile phone, and give access to her family.

- User-centered discovery of services:

  * Users can search for services based on friendly criteria such as "I want to see services published by my friends," or "I want to see services in this shopping center."

  * Users can do explicit discovery of services, e.g. by reading an RFID tag or taking a photo of a bar code.

**Research questions**

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

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

**Expected deliverables**

- Scenarios for user-centered service discovery.
- Architecture and design (in UML 2.0) for UbiCollab service discovery subsystem.
- Extensions, in form of Java code, to existing service discovery subsystem in UbiCollab.

- Extensions, in form of Java interfaces, to existing service discovery APIs in UbiCollab.
- Implementation (in Java) and testing (in JUnit) of user-initiated discovery plugins for RFID, Bluetooth and optical barcodes.
- GUI (In Java) for allowing users to control UbiCollab service discovery subsystem.

Note that UbiCollab is open source. This means your contributions in terms of architecture, design and code will be submitted to an open source project under Apache 2.0 terms.

## A.2 Scenario

***Collaborative care arena scenario from UbiLife***

*Knut is an early retired man in his 50s with a chronic heart disease. He is married to Inger. He lives at home but is in need of frequent medical care. Marie is Knut's contact nurse at the hospital. Marie is usually responsible for 10-15 patients in Knut's situation. In order to avoid daily hospital visits for the patients, the hospital is using an e-care system. The hospital personnel, all patients and their closest relatives have received a connected mobile care device (MCD). On this morning, Marie arrives at the hospital. After taking a cup of tea, she goes to an e-Consulting station in the closest room (e-Consulting stations are widely deployed at the hospital). She touches a tag on the station using her MCD's tag reader. The station immediately shows her list of patients. Each patient has his/her own dedicated Collaborative Care Arena (CCA). The list of CCAs includes a small photo of each patient, a date showing the time of last consultation and home visit, and a color bar showing status of measured data. She notices that Knut's CCA has a shade of yellowish green instead of saturated green. She touches Knut's CCA. At this time Knut is at home reading his newspaper while eating his breakfast. He*

*hears a knocking sound from the e-Consulting station in the kitchen. Marie's photo is shown on the screen. He touches the station using his MCD, which activates the CCA and a video conference is set up between Knut and Marie. They have a short chat and Marie brings up the CCA data window by touching an icon. The data window shows a small map of Knut's house, with a red dot showing that he is in the kitchen. Together they review the data collected from medical sensors in the house. Knut informs Marie that he has received a new life jacket (a medical vest with integrated sensors). An icon below the CCA window indicates that the life jacket is ready to be used. Knut drags the icon into the CCA. Marie asks Knut to put on the jacket. Knut goes to the bedroom where the jacket is. The video conference is automatically moved to the e-Consulting station in the bedroom and is changed to an audio conference due to Knut's privacy preferences. After Knut has the jacket on, Marie clicks on a couple of icons in the CCA window and a fresh measurement from the jacket is taken. After a short discussion Marie decides that there is nothing critical. She reminds Knut to take his medication and says goodbye. After breakfast, Knut's wife Inger goes to work and Knut decides to take his morning walk. Knut is active in an online local community initiated by a group of heart patients who compete on walking longest. He can see in his CCA that some members are already walking in the park. He leaves home wearing his life jacket and his step teller. While in the park, Knut feels a pain in his chest. As this happens, Marie's MCD makes an emergency alert. Marie, not seeing any e-Consulting station nearby, uses her mobile phone to view Knut's CCA and start a phone conversation with him. The color is red and Marie decides to click on the CCA alert icon. The MCDs of Alison (the heart specialist) and Knut's wife Inger start alerting. Alison is already at an e-Consulting station and gets Knut's CCA on the screen. A voice conference between Marie, Alison and Knut is set up immediately (using Knut's and Marie's mobile phones). After a short talk with Knut (who is sitting on a bench in the park) Alison decides to involve ambulance personnel and bring Knut to the hospital. She touches CCA's ambulance icon. Arne (ambulance*

*driver) is available and closest to the park. His MCD alerts him. The voice conference is extended to Arne's mobile phone. At the same time Arne gets Knut's CCA on his screen inside the ambulance. The CCA shows the latest data, including Knut's location in the park. By this time Marie has had a phone conversation with Inger, and she is on her way to the hospital. Knut pushes an icon on his CCA and a phone call between him and Inger is set up. He talks to Inger while he is waiting for the ambulance.*

# Appendix B

# UbiCollab Runtimes

This appendix defines the running environment for the UbiCollab platform used during the evaluation. Since at the present time platform setup wizards have not been elaborated yet, this list of components can be used if the reader would test our platform on his/her personal device. Components are grouped by functionality area and a short description is provided.

## B.1 Runtime components

**Java Virtual Machine:** IBJ J9 6.2, CDC 1.1, Foundation Profile 1.1

**OSGi:** R4 Specifications implementation from Eclipse Equinox project

**OSGi Bundles:**

`org.eclipse.osgi v. 3.2.2`
`org.eclipse.osgi.services v. 3.1.2`

`org.eclipse.equinox.cm v. 3.2.0`
`org.eclipse.equinox.common v. 3.2.0`
`org.eclipse.equinox.preferences v. 3.2.1`

```
org.eclipse.equinox.registry v. 3.3.1
```

**Eclipse Runtimes:**

Shared components used in the Eclipse Foundation projects:

```
org.eclipse.core.commands v. 3.2.0
org.eclipse.core.contenttype v. 3.2.0
org.eclipse.core.expressions v. 3.2.2
org.eclipse.core.jobs v. 3.2.0
org.eclipse.core.runtime v. 3.2.0
org.eclipse.core.runtime.compatibility.auth v. 3.2.0
```

**eRCP:**

```
org.eclipse.ui      v. 1.2.0
org.eclipse.ercp.ui.workbench   v. 1.2.0
org.eclipse.ercp.jface v. 1.0.2

org.eclipse.ercp.xml v. 1.0.2
org.eclipse.ercp.xmlParserAPIs v. 1.0.2
```

**eSWT:**

```
org.eclipse.ercp.swt v. 1.0.2
org.eclipse.ercp.swt.win32 v. 1.2.0
  (swt native implementation for Windows)
org.eclipse.ercp.swt.wm2003 v. 1.0.2
  (swt native implementation for Windows Mobile)
```

**WebServices:**

```
axis-osgi v. 1.4.0
http\_all v. 2.0.0
  (apache web server implementation)
javax.servlet v. 2.4.0
log\_all v. 2.0.0
```

```
org.apache.commons.logging v. 1.0.4
org.apache.xerces v. 2.8.0
  (xml parser used by Axis)
```

## B.2 Tools Used for Development

- Eclipse Ganymede[1] 3.4.2: Java Coding, Debugging.
- Instantiations SWTDesigner[2]: GUIs visual editing
- Soyatec eUML2[3]: UML modeling, report graphs
- Apache Axis 1.4 WSDL2Java emitter[4]: Development of Java stubs for WebServices
- Retroweaver 2.0.7[5]: Java 1.5/1.6 to Java 1.4 recompiling

## B.3 Compatibility of Code

Several components have been developed in $Java >= 1.5$. This implies that the code utilizes newer language specific features, such as generics, that are not compatible with older Java versions. Because currently does not exist a reliable CDC JVM which supports a $JDK > 1.4$ and thus these newer language specific features and syntax, the existing code either needs to be rewritten or adapted to be in conformance to Java 1.3. Retroweaver is an open source library that enables the use and advantages of newer Java language features on legacy versions of JVMs. Specifically, it compiles 1.5 source to 1.4 bytecode, which thus can be run with the JVM implementations we

[1]Eclipse Ganymede: http://www.eclipse.org/downloads/
[2]Instantiations eRCP/SWT Designer: http://www.instantiations.com/ercpdesigner/
[3] Soyatec eUML2: http://www.soyatec.com/euml2/
[4]Apache Axis 1.4: http://ws.apache.org/axis/
[5]Retroweaver project: http://retroweaver.sourceforge.net

used in our tests. Thanks to this, code may be written using more advanced language syntax, and still be able to run on mobile devices.

# Appendix C

# Devices Specifications

This appendix show technical specifications of the devices employed for testing during the research and the evaluation. The name of the device in UbiCollab terminology is reported next to device's brand and model.

# C.1 HTC Touch HD - UbiNode

| Feature | Description |
|---|---|
| Dimension | 115 x 62.8 x 12mm |
| Weight | 147g. |
| Display Type | Capacitive TouchScreen |
| Display Size | 3.8” 800x480px |
| WLAN | 802.11b/g Bluetooth 2.0 |
| Processor | ARM 528Mhz |
| RAM | 288Mb |
| Operative System | Windows Mobile Professional 6.1 |
| Other Features | 5MPX Camera, GPS, Accelerometer |

## C.2 IDBlue RFID Bluetooth Reader

DBlue is a 13.56 MHz RFID reader, in the shape of a pen with dimension 112mm x 25mm x 16mm and a weight of 50 grams. It uses Bluetooth technology to wireless link up with most Bluetooth compliant devices, such as PCs, PocketPCs and SmartPhones.

### C.2.1 Device Features

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

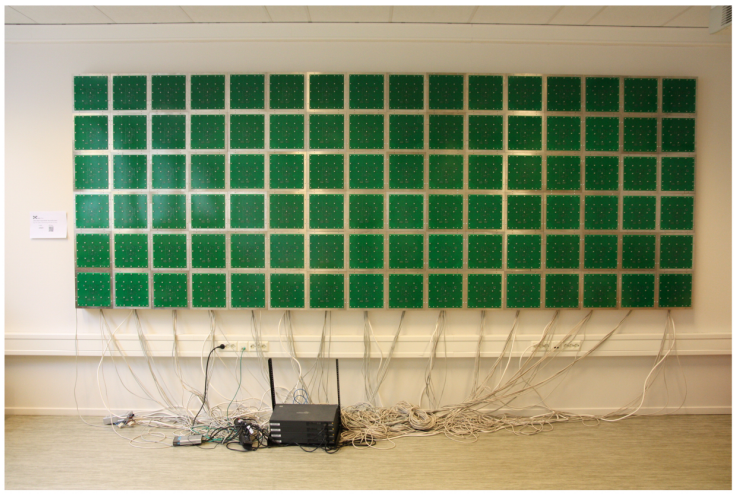
  - Read data
  - Write data

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

## C.3 Asus R2H TabletPC - SharedScreen


| Feature | Description |
|---|---|
| Dimension | 234,2 x 133 x 28 mm |
| Weight | 960g. |
| Display Type | Capacitive TouchScreen |
| Display Size | 7” 800x480px |
| WLAN | 802.11b/g Bluetooth 2.0 |
| Processor | Intel Celeron-M ULV 900Mhz |
| RAM | 512Mb |
| Operative System | Windows XP SP2 |
| Other Features | 1.3MPX Camera, GPS, Fingerprints reader |

## C.4 IDI Open Wall - Shared Screen



| Feature | Description |
|---|---|
| Display Type | LED Screen, grayscale (99 shades of orange) |
| Display Size | 480x180cm 80x30px |
| Others | Java APIs |

# Bibliography

[1] M. Divitini and B. Farshchian, "Ubicollab architecture white paper," 2007.

[2] K.-S. Johansen, "User-centered and collaborative service management in ubicollab - design and implementation," Master's thesis, 2007.

[3] M. Weiser, "The computer for the 21st century," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, 1999. 10.1145/329124.329126.

[4] K. Schmidt and L. Bannon, "Taking cscw seriously: Supporting articulation work," *COMPUTER SUPPORTED COOPERATIVE WORK*, vol. 1, pp. 7–40–7–40, 1992.

[5] B. A. Farshchian and M. Divitini, "Ubicollab: Improving collaboration with location services," 07/2005 2005. 417- 420.

[6] P. Kruchten, "Architectural blueprints: The "4+1" view model of software architecture," *IEEE Software*, vol. 12, no. 6, pp. 42–50, 1995.

[7] D. Saffer, *Designing Gestural Interfaces: Touchscreens and Interactive Devices*. O'Reilly Media, Inc., 2008.

[8] B. Shneiderman, "Direct manipulation: A step beyond programming languages," *Computer*, vol. 16, no. 8, pp. 57–69, 1983.

[9] M. A. Lebedev and M. A. L. Nicolelis, "Brain-machine interfaces: past, present and future," *Trends in Neurosciences*, vol. 29, no. 9, pp. 536–546, 2006. 10.1016/j.tins.2006.07.004.

[10] K. A. G. Olsen, "Distributed session management, session mobility and transfer in ubicollab," master, Norwegian University of Science and Technology (NTNU), 06/2008 2008.

[11] Microsoft, "Windows mobile team blog." http://blogs.msdn.com/.

[12] SAP, "Interaction design guide for touchscreen applications."

[13] K. Dandekar, B. I. Raju, and M. A. Srinivasan, "3-d finite-element models of human and monkey fingertips to investigate the mechanics of tactile sense," *Journal of Biomechanical Engineering*, vol. 125, no. 5, pp. 682–691, 2003. 10.1115/1.1613673.

[14] F. Tournier, "Java mobility roadmap," tech. rep., 2009.

[15] "Osgi alliance, specifications." http://www.osgi.org/specifications/.

[16] "Eclipse equinox." http://www.eclipse.org/equinox/.

[17] "Knopflerfish osgi." http://www.knopflerfish.org.

[18] "Apache felix." http://www.apache.com/felix/.

[19] "Concierge osgi." http://concierge.sourceforge.net.

[20] "Eclipse rcp." http://www.eclipse.org/ercp/.

[21] "eswt mobile extension programming guide," tech. rep.

[22] "Jalimo project." http://www.jalimo.org.

[23] "Eclipse public license 1.0." http://www.eclipse.org/epl/.

[24] B. Marchal, "Working xml: Define and load extension points," 2005.

[25] E. Mok and G. Retscher, "Location determination using wifi fingerprinting versus wifi trilateration," *J. Locat. Based Serv.*, vol. 1, no. 2, pp. 145–159, 2007.

[26] I. Satoh, "A location model for ambient intelligence," in *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, (Grenoble, France), pp. 195–200, ACM, 2005. 10.1145/1107548.1107598.

[27] D. L. Lee and Q. Chen, "A model-based wifi localization method," in *Proceedings of the 2nd international conference on Scalable information systems*, (Suzhou, China), pp. 1–7, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[28] S. Benford, A. Crabtree, M. Flintham, A. Drozd, R. Anastasi, M. Paxton, N. Tandavanitj, M. Adams, and J. Row-Farr, "Can you see me now?," *ACM Trans. Comput.-Hum. Interact.*, vol. 13, no. 1, pp. 100–133, 2006. 10.1145/1143518.1143522.

[29] M. Youssef, M. Mah, and A. Agrawala, "Challenges: device-free passive localization for wireless environments," in *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, (Montréal, Québec, Canada), pp. 222–229, ACM, 2007. 10.1145/1287853.1287880.

[30] Q. Fu and G. Retscher, "Using rfid and ins for indoor positioning," 2009. Location Based Services and TeleCartography II.

[31] S. A. Karsten Schmidt, "The sap eclipse story," tech. rep., 2008.

[32] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "Building, deploying, and monitoring distributed applications with eclipse and r-osgi," in *eclipse '07: Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*, (New York, NY, USA), pp. 50–54, ACM, 2007.