FEDERAL UNIVERSITY OF RIO GRANDE DO SUL
INFORMATICS INSTITUTE
BACHELOR OF COMPUTER SCIENCE

BRUNO ROMEU NUNES

An automation system for ubiquitous computing

Graduation Thesis

Prof. Dr. Claudio Geyer
Advisor

M. Sc. Valderi Leithardt
Coadvisor

Porto Alegre, October 2013

CIP – CATALOGING-IN-PUBLICATION

Nunes, Bruno Romeu

An automation system for ubiquitous computing / Bruno Romeu Nunes. – Porto Alegre: Graduação em Ciência da Computação da UFRGS, 2013.

63 f.: il.

Graduation Thesis – Federal University of Rio Grande do Sul. BACHELOR OF COMPUTER SCIENCE, Porto Alegre, BR–RS, 2013. Advisor: Claudio Geyer; Coadvisor: Valderi Leithardt.

1. Ubiquitous computing. 2. Context-aware computing. 3. Mobile computing. 4. Localization. 5. Android automation. I. Geyer, Claudio. II. Leithardt, Valderi. III. Título.

ACKNOWLEDGMENTS

I am very thankful to my family for all of the education and support they have provided me since I was born. They helped me define who I am, and they will have part on everything I ever achieve in life.

Thanks UFRGS and Informatics Institute, for the all of the opportunities I had while studying here, and for the incredible quantity and quality of knowledge that I could aggregate in these last years. Specifically, thanks to all of the motivated professors that taught me how interesting it is to learn.

Thanks professor Geyer, Valderi and Guilherme for all the help on the development and conception of this project.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

GCM	Google Cloud Messaging
API	Application Programming Interface
SDK	Software Development Kit
GPS	Global Positioning System
RFID	Radio-Frequency Identification
NFC	Near Field Communication
NDEF	NFC Data Exchange Format
UML	Unified Modeling Language
IEEE	Institute of Electrical and Electronics Engineers
VM	Virtual Machine
OS	Operating System

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Since the emergence of the first computers, they have constantly become smaller and more powerful, and also have spread over our everyday. Nowadays, they are capable of controlling diverse features of the environment where they are located.

Inspired by the concepts of ubiquitous computing and context-aware computing, this work proposes the model of a system that integrates these multiple devices and performs changes over their functionalities, responding to changes in the context. The goal is to automatically and proactively adapt the environment to the preferences defined for the present users.

A prototype was implemented based on the model created by us. The prototype consists of a server, responsible for the management of the client applications, and a client application for the Android platform, which automates the settings of some functionalities of the device. Features like the Bluetooth, Wi-Fi or the device's volume are changed in response to changes on the user's location, conforming with rules defined for the environment where he is located.

Keywords: Ubiquitous computing, context-aware computing, mobile computing, localization, Android automation.

RESUMO

Desde o surgimento dos primeiros computadores eles têm se tornado constantemente menores e mais potentes, e aparecido em maior número em nosso cotidiano. Atualmente eles são capazes de controlar diversas características do ambiente no qual se situam.

Inspirado pelos conceitos de computação ubíqua e de computação ciente de contexto, este trabalho propõe o modelo de um sistema que integra estes múltiplos dispositivos e executa alterações em suas funcionalidades em resposta a mudanças de contexto. O objetivo é adaptar o ambiente às preferências definidas para os usuários presentes, de maneira automatizada e proativa.

Um protótipo foi implementado com base no modelo que criamos. Este protótipo é composto por um servidor, responsável pelo gerenciamento das aplicações clientes, e de uma aplicação cliente para a plataforma Android, que automatiza a configuração de algumas funcionalidades do dispositivo. Funcionalidades como Bluetooth, Wi-Fi ou o volume do dispositivo são alteradas em resposta a mudanças na localização do usuário, atendendo a regras definidas de acordo com o ambiente onde ele se encontra.

Palavras-chave: Ubiquitous computing, context-aware computing, mobile computing, localization, Android automation.

1 INTRODUCTION

Section 1.1 talks about the motivation to develop this work. Section 1.2 presents what we expect to achieve in the development of this project. Section 1.3 uses examples to illustrate real world applications of our system. Section 1.4 depicts the structure of this thesis.

1.1 Motivation

Computers started out as big and expensive machines, with the purpose of helping with heavy calculations, and are now cheap, small and powerful enough to be embedded everywhere we need them. Thanks to this progress, we are currently living the process of proliferation of the computers and their fade to the background, as predicted by Mark Weiser back in 1991 (Weiser 1991).

Every day more and more computers are added to objects that are already part of our lives. Televisions now have complex operating systems that support online video streaming, video conferences and internet surfing; refrigerators include LCD screens and run custom applications; door locks are controlled by computers; cars include complete computers that are used to control functions of the car, navigate, or even to provide entertainment.

Besides these examples of products that have already been developed, there are still many others being built regularly from both academic and commercial research as a proof of concept. They embed sensors, communication modules, or even entire computers into regular objects to give them intelligence and connectivity. Although at this time they are only ideas, it is safe to predict that a lot of them will eventually be absorbed by society.

There is still the possibility of connecting objects by ourselves. Open source hardware, such as the Arduino microcontroller (Arduino 2013), are cheap and capable enough to allow us to programmatically control at least the basic functionalities of electronics, like switching on and off.

Nearly every object is becoming equipped with sensors and somehow getting connected to the internet, creating the "Internet of Things" (Ashton 2009).

Smartphones are becoming essential in people's lives; even though somewhat recent in existence, they are already deeply pervaded over the world. They are powerful computers that are always accompanying their owners, and are also very personal: users customize them and fill them with personal information and data. Together with their reduced size and constant connectivity, these features make such a device perfect for use as an identification of its owner, allowing the development of context-aware computing around it.

Although this proliferation of computers has made it easier for us to execute tasks, the way we interact with them could still be greatly improved. By now we still have to

explicitly send them some sort of command to let them know we want an action to be executed. With this work, we want to integrate these computers, creating a system that preemptively adapts the environment to the preferences of the users, without interaction from them.

1.2 Objectives

The final goal of this project is to create the basis for an expansible system that manages ubiquitous computing environments. The system is intended to integrate multiple computers that are responsible for the control of features of the environment, and manage these features to fit the preferences defined for the context.

Using location sensors, the system will be aware of the presence of users in the environment, and will adapt it to best match the preferences of the people that are currently present.

These preferences are defined in terms of features of the environment, which are characteristics that can be changed by our system. Features can include the following, in addition to many more: the temperature of the room, the state of the lighting system, or the Bluetooth status of devices in the environment.

The features and environments that the system supports will be expansible, depending solely on the integration of new clients that manage specific features on specific environments. The system aims to easily integrate new clients to make its expansion simple.

In this study, a model of the system will be designed, and a prototype will be developed. The prototype consists of an implementation of the complete model, focusing on an Android application.

We will take advantage of one of the most powerful features of modern smartphones: the presence of multiple embedded sensors. The Android application will use multiple location technologies to detect the presence of the user in the covered environments. The application will also turn the device into part of the environment by automatically managing functions of the smartphone according to preferences defined for the environment.

1.3 Application Scenarios

With the integration of multiple devices the system will be able to control different features of the environment. Figure 1.1 illustrates the elements of the environment that are changed in response to the user's presence in the following scenario:

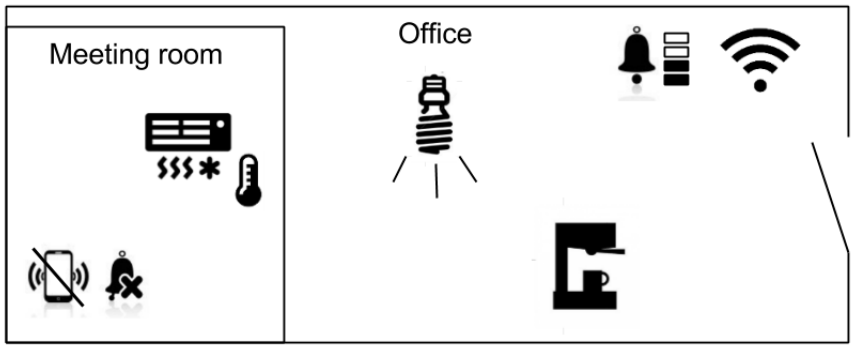


Figure 1.1: Application Scenario

- Albert works at an IT company. Every day when he arrives at work his phone's Wi-Fi is turned on, volume is set low and the coffee maker starts making his favorite coffee. If he is the first to arrive, the lights are turned on. Today he has a video meeting in the room 205 from 11:00 to noon. He enters the room and the air conditioning system is turned on at his preferred temperature. Right before his meeting begins, his phone automatically sets to silent, and his vibration alert is turned off. When the meeting is over, his phone is set back to his workplace settings, and the air conditioner is turned off. When he leaves work, his phone's settings are back to the way they were when he arrived. If he is the last one to leave, the lights of the office are turned off.

The prototype developed in this work will focus on the integration of Android devices to the system. The application will be able to automatically change settings on the smartphones of users based on their location. Some scenarios that exemplify the use of the system are:

- John, willing to watch his favorite director's last movie, buys a ticket to Saturday's 14h session at the usual movie theater. At 13:30 he heads to the cinema and, upon arrival, uses his smartphone to gain access to the theater. His phone's volume is set to medium and his vibration alert is enabled as he walks to his seat. When the movie is about to start, his phone is put into silent mode and the vibration alert is disabled. When it ends, John leaves the theater and the phone is set back to its previous settings.

- Andrew studies at his city's university. Every time he enters the university's campus, his Wi-Fi is turned on so he can connect to the university's network. When he is at the building where he studies, his phone's volume is set to low, and when he enters his class room, it is set to silent mode automatically. When he is at his car, his phone's Bluetooth is turned on to stream audio to his car's speakers. As he leaves each environment, these settings are changed back to their previous state.

1.4 Organization

This paper as organized as follows: Chapter 2 introduces the concepts and technologies studied and used in this work; Chapter 3 presents researched works that relate to our prototype; Chapter 4 explains how the problem was modeled in order to achieve our goals; Chapter 5 details how the prototype was implemented; Chapter 6 explains the tests we executed to validate the prototype implementation; and Chapter 7 presents the conclusions we have come to in the development of this project, and future works that can improve and expand the system.

2 CONCEPTS AND TECHNOLOGIES

This chapter presents an overview of the concepts and technologies that were studied and used on the development of this work. Section 2.1 and 2.2 present the concepts of ubiquitous computing and context-aware computing, respectively. Section 2.3 discusses and compares the location technologies used on the prototype. Section 2.4 introduces Android, the open source operating system on which the prototype client application runs.

2.1 Ubiquitous Computing

Ubiquitous computing is a term coined by Mark Weiser during his work at the Computer Science Laboratory at the Xerox PARC (Palo Alto Research Center). Between 1988 and 1994, as the head of the laboratory, he wrote or co-wrote dozens of publications that defined key concepts in the area.

In one of his early publications (Weiser 1991), he talks about the work of his team at PARC and introduces the concept of ubiquitous computing. He states that in the future, computers would not be limited to the traditional desktop and notebook models; they would appear on multiple different forms, depending on their main purpose. Computers' presence would become widely spread across every environment, and our interaction with them would become unconscious.

According to Weiser, that is the trend for technologies that bring real improvements on people's capabilities and examples are there to prove it: in less than a century, the electric motor turned from a factory tool to an invisible and omnipresent engine, which is elemental to uncountable staple machines. Ubiquity would then be the next wave in computing, too, and computers would disappear in the background of our lives.

Bringing computers to the background is not easy, though; it would require a shift of the way we interact with them. The traditional interaction demands us to make the computer the center of attention. The interaction should instead be imperceptible, allowing people to just focus on what is really important (Weiser 1993). That is what would allow computers to achieve their full potential.

2.2 Context-Aware Computing

Context-aware systems are those that adapt to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time (Schilit et al. 1994). Such systems can examine the computing environment and react to changes to the environment.

Schilit et al. investigate context-aware applications categorizing them into four groups, as shown in table 2.1. The categories group applications by whether the task at hand is getting information or carrying out a command, and whether it is effected manually or automatically.

	manual	automatic
information	proximate selection & contextual information	automatic contextual reconfiguration
command	contextual commands	context-triggered actions

Source: (Schilit et al. 1994)
Table 2.1: Context-Aware Software Dimensions

Proximate selection is a user interface technique that uses location information and emphasizes objects that are nearby.

Automatic contextual reconfiguration is the process of adding, removing or altering the connections between components of the system upon changes in the context.

Contextual information and commands aim to exploit the predictability of people's actions based on the situation. Such applications produce different results for the same action on different contexts.

Context-triggered actions are rules used to specify how context-aware systems should adapt, using information about the context to trigger commands. Applications that fit in this group have actions invoked automatically according to previously specified rules.

This work is focused on commands, and therefore it fits better in the last two groups. Specially, we focus on automation, and make use of context-triggered actions to adapt the system to the context.

2.3 Localization

Location is an essential knowledge in ubiquitous systems. It is necessary to know where the users are and what objects are around them to act accordingly. Various technologies have been developed that can be used to locate objects spatially. These technologies have different focuses, use different techniques and can generate different results.

Hightower and Borriello (Hightower e Borriello 2001) explain the three principal techniques used for automatic location sensing. The most used location systems rely on one or more of these techniques:

- *Triangulation* - Can be divided in the sub-categories *lateration*, when using distance measurements, or *angulation*, when using angle or bearing measurements. It is basically the application of trigonometry to determine the position of the object based on the location of reference points and the distances or angles between them and the object.

- *Scene analysis* - Compares features of a scene with a predefined dataset (*static* scene analysis) or with successive scenes (*differential* scene analysis) to infer the location of the observer or of objects in the scene.

- *Proximity* - Uses a physical phenomenon with limited range to determine when an object is near a known location.

In another work, Hightower and Borriello propose a taxonomy to evaluate and compare location systems (Hightower e Borriello 2001), which can be summarized in the following topics:

- *Physical position and symbolic location* - A location system can provide *physical* positions that actually locate the object in the space using some coordinate system; or *symbolic* location, which is an abstract idea of the object's location.
- *Absolute versus relative location* - *Absolute* locations use a shared reference grid for all objects, while *relative* locations use a reference relative to the location system object.
- *Localized location computation* - The computation of the location is done locally by the object that is being located, such that the location system does not need to know the position of the object.
- *Accuracy and precision* - *Accuracy* relates to the grain size, or the exactitude of the position. *Precision* determines how often we can expect to get that accuracy.
- *Scale* - The *scale* is a measurement of the system's coverage area and number of objects the system can locate, per unit of infrastructure and time.
- *Recognition* - Refers to the ability of the system to recognize the objects located in it.
- *Cost* - A general measurement of the time, space and capital costs of using the location system.
- *Limitations* - Limitations of the system and requirements for proper functioning.

In the next subsections, we will introduce the location techniques that were integrated into the system implemented in this work. We then compare them using the taxonomy that was just presented.

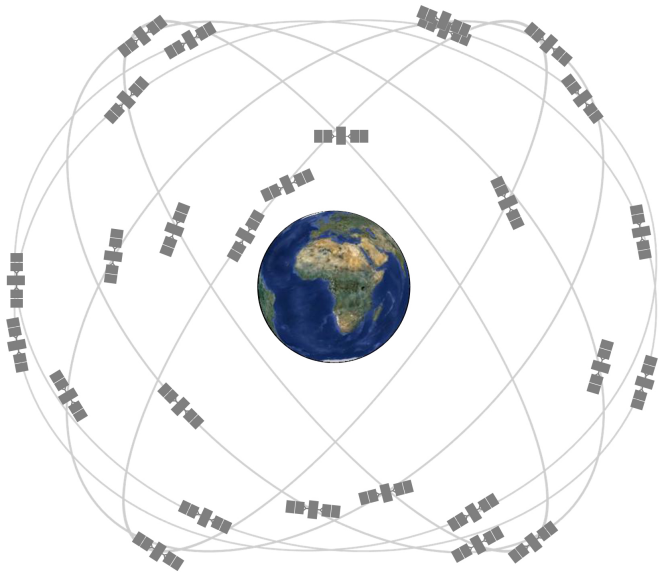
2.3.1 GPS

The Global Positioning System (GPS) is a satellite-based navigation system. It is composed of three segments: Space, Control and User.

The Space segment is composed of a constellation of at least 24 satellites, 20,200Km above the earth with 12 hours period, that are constantly broadcasting data through radio signal. As shown in Figure 2.1, the satellites are arranged in six orbital planes with four satellites each, ensuring that virtually any point on the planet is in the line of sight of at least six satellites at any given time (Center 2013). The data transmitted can be divided in two parts. The first contains the satellite's position and a very precise timestamp. The second contains general data about the state of the whole constellation.

The Control segment consists of monitor and control stations in the earth, responsible for keeping the system operational. They track all of the GPS satellites, maintaining them on their orbit. They also keep their clocks synchronized and update their navigation messages. Both the Space and the Control segments are developed, maintained and operated by the U.S. Air Force (Navigation e Timing 2013).

Finally, the User segment encompasses the receiver equipment, which gathers the data sent from the satellites, and use it to determine its position on the earth. The position is



Source: www.gps.gov
Figure 2.1: The GPS satellites constellation

defined by measuring the distance between the receiver and at least three satellites, for latitude and longitude location, or four satellites, to determine altitude.

The major downside of the GPS system is that the frequency of the signal impedes the information from passing through most solid objects, like buildings or mountains. This restrains its functioning indoors, or in areas where the sky is not visible.

2.3.2 RFID

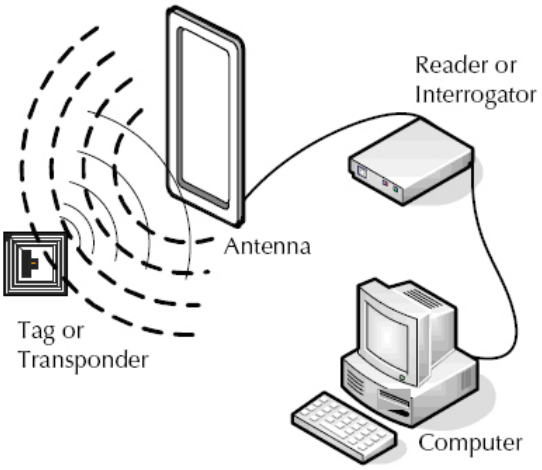
RFID stands for Radio Frequency Identification, and consists of the use of radio-frequency electromagnetic fields to identify objects. A typical RFID system is composed of RFID tags and RFID readers.

RFID tags (also known as RFID labels) contain non-volatile information electronically stored in a microchip and an antenna for receiving and transmitting radio-frequency signal. Some tags, called active tags, are also equipped with a battery that allows them to actively send their waves. Passive tags have no power source, and can only respond to readers using energy from the electromagnetic field they create. Tags can be read-only, with its value assigned by the time of its creation; or read-write, with the possibility of changing its value multiple times (Yang et al. 2013).

RFID readers can also be active or passive. Active readers interrogate passive tags and emit electromagnetic energy to power the tag while it is sending information. Passive readers are constantly ready to receive signals from active tags. Readers then demodulate the signal received from the tags and make the information available for processing by the system.

The RFID tags usually store only a unique identifier that is associated with the object the tag is attached to. The complete information about each object stays in a database that is queried to collect or add relevant information when an object ID is read (INFO 2013).

RFID systems are largely used for the tracking of objects in industry and transportation systems. Since passive tags are inexpensive and have a long lifetime, they can be attached to the body of the object intended to be tracked and RFID readers can identify and locate that entity. This method's major advantage over optical methods, such as bar



Source: www.epc-rfid.info/rfid
Figure 2.2: An RFID system

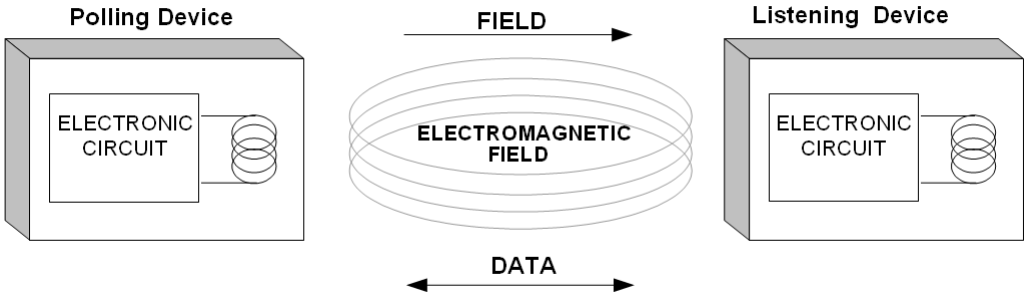
codes, is that RFID tags are readable through small obstacles. This facilitates the reading of its content without the need to be in the line of sight of the reader.

Other fields where RFID is currently being used are: animal chipping, in farms for livestock tracking and in pets for contact information; human chipping, for emergency contact information and medical history; smart cards, for access management; vehicles tagging, for electronic toll collection; and passports, for security increase and storage of relevant information.

2.3.3 NFC

NFC, short for Near Field Communication, is a technology for short range wireless communication. Built on top of the RFID technology, it consists of a set of standards designed to allow secure data exchange between two NFC-compatible devices. A NFC communication is established by touching the devices together or bringing them a few centimeters from each other. It operates at around 13.56MHz, offering a data transmission rate of up to 424kbit/s (Schwarz 2011).

As opposite to RFID, NFC allows two-way communication, what means that two NFC enabled devices can talk to each other alternating the roles of active and passive agent. This feature makes it possible for applications running on each of those devices to actually communicate, sharing information in real time instead of only pre-stored data.



Source: (Schwarz 2011)
Figure 2.3: NFC data transmission

The NFC technology began to be defined in 2004, when Nokia, Philips and Sony established the NFC Forum (Forum 2004). Since the first NFC-enabled mobile phone was launched, in 2006, the number of devices that support the technology has been growing. According to a research by Juniper Research in 2011, about 20% of the smartphones will have NFC by 2014 (Juniper 2011).

The combination of short range and two-way communication makes NFC the ideal technology for more secure systems. Currently, the major use of NFC is in mobile payments, authentication for secure access and ticketing. The presence of NFC in mobile devices is in large expansion

NFC is slower than other wireless communication technologies such as Bluetooth or Wi-Fi, but it is much simpler to setup. This fact has raised yet another use for it: two NFC-enabled devices can touch each other and use NFC to pair or set up a network between them, then use this connection to exchange heavy data.

Like RFID, NFC can also act as a means of communication between active devices and passive tags; these are called NFC tags. NFC tags can store up to 32KB of data, depending on its type, and can also be read-only or read-write.

2.3.4 Wi-Fi Positioning System

The Wi-Fi is a wireless technology based on the IEEE 802.11 standards (Alliance 2013), that operates in the 2.4 and 5 GHz radio bands. These bands are "license-free", i.e. anyone can use them without government license. For that reason, Wi-Fi became popular in the form of wireless local area networks (WLAN), used for personal and business wireless communication. Although not originally created with location purposes, the rapid growth of wireless access points in urban areas created an opportunity to use them to generate location data.

Big companies created and started to populate large databases with information about the access points available in a given area and their locations. Devices collect the mandatory access control (MAC) address and the signal strength of networks that publicly broadcast their service set identifier (SSID), and send them to servers together with the position they got from another location methods, like GPS. With that information, it is possible to get the proximate location of a device by knowing the WLANs in range and their signal strength (Milette e Stroud 2012).

Google is one of those companies, and it uses Android devices as helpers. When the user enables Google's location services as a source of location data on his device, the system starts collecting information about the visible Wi-Fi access points and their signal strength, and uses them to get positions from Google's servers. The device will, in exchange, send anonymous location data that helps improve the service and keep up-to-date data.

2.3.5 Cell IDs

The use of Cell IDs for localization is based on the same principle of the use of Wi-Fi access points. Cellular devices must always be in contact with a cell tower, and each cell tower has its unique ID. By mapping the location of each cell tower and knowing the ID of the one the device is connected to, it is possible to estimate the device's position.

This method is also used by Google's location services. They collect information sent from Android devices that know their position and the towers near them. This information is used to map the cell towers by their IDs. With enough towers mapped, it is then possible to project the position of the devices that only know the cell tower ID in which they are

currently connected (Milette e Stroud 2012).

2.3.6 Evaluation of location systems

Table 2.2 compares the location systems using the taxonomy presented earlier in this section. Data comes from the sources cited on the previous sections.

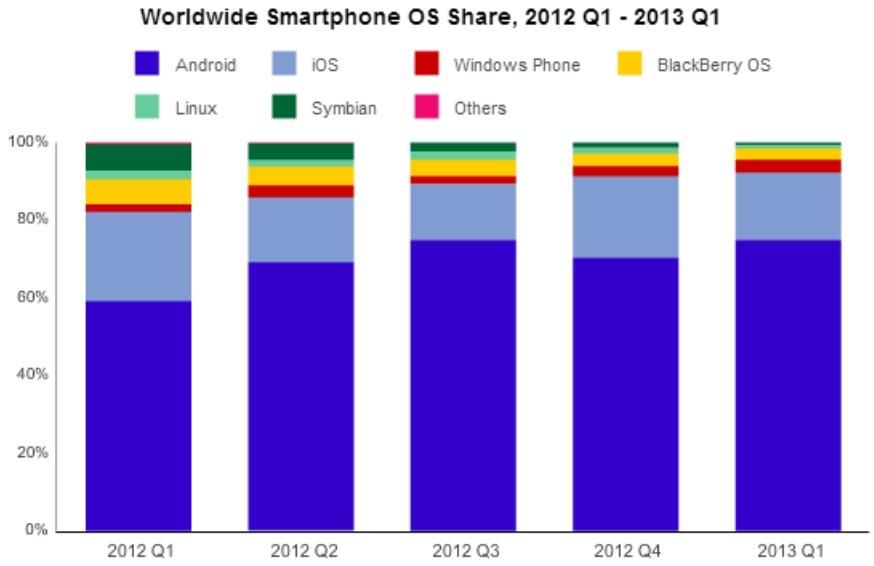
2.4 Android

Android is an open source operating system for mobile devices, built to enable the development of applications that take full advantage of all the features that handsets can offer (Alliance 2007). Its development started in 2003 by Android Inc. in Palo Alto, California. In 2005, Google bought the company and started working on the system in cooperation with handset manufacturers and mobile network operators. The system was unveiled in 2007, with the founding of the Open Handset Alliance, a consortium of 84 technology and mobile companies aiming to develop open standards for mobile devices (Alliance 2007).

2.4.1 Market Share

Android is the most used mobile operating system worldwide, followed by Apple's iOS. Together, they account for over 92% of smartphone shipments during the first quarter of 2013, according to IDC (IDC 2013).

Table 2.3 compares the number of smartphones of each operating system shipped worldwide during both the first quarter of 2012 and the first quarter of 2013. Android shows a growth of 79.50%, jumping from 59.10% of the market share to 75%. This is 4.33 times more than iOS, which is in second place. Figure 2.4 shows the market share for the same period for each quarter.



Source: IDC Worldwide Quarterly Mobile Phone Tracker, May 2013 (IDC 2013)
Figure 2.4: Worldwide Smartphone OS Share, 2012 Q1 - 2013 Q1

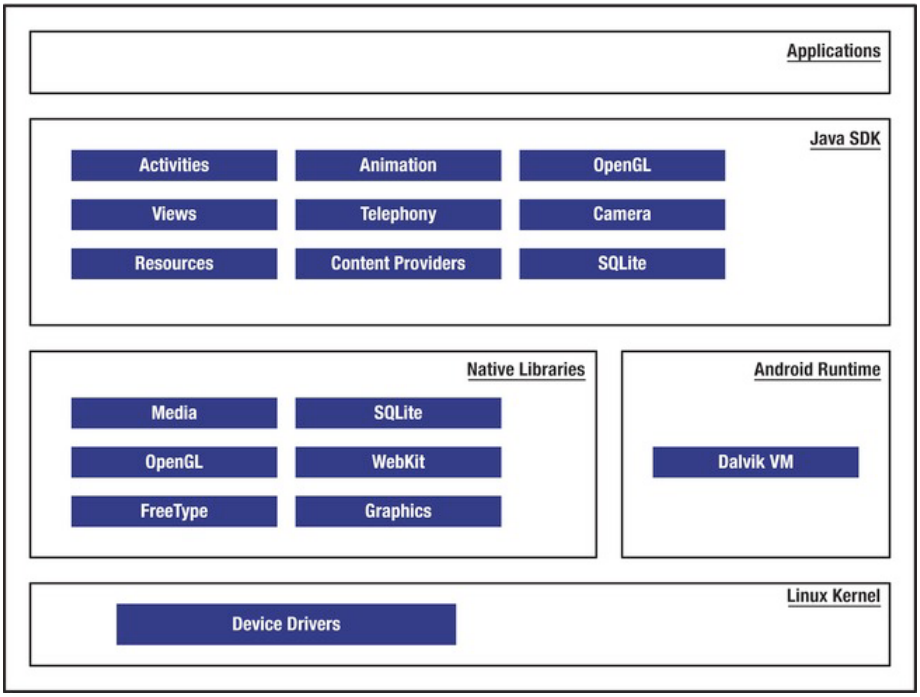
2.4.2 Versions

The first Android phone was the HTC Dream, launched in October 2008 with the Android version 1.0 (API level 1). Since then, 16 new versions have been launched along with new APIs that expose the new features for application developers.

The wide variety of devices that run Android causes the system to suffer from fragmentation. That means the latest versions of the system take some time to reach the majority of the active devices, which is why it is necessary for the developers to use older APIs to support the maximum number of devices. Table 2.4 lists the Android versions and their distribution in the current active devices (data from July 2013) (any versions with less than 0.1% distribution are not shown).

2.4.3 Architecture

Android is built on top of a Linux kernel, and its structure can be divided into a few layers. Figure 2.5 helps better explain the architecture of the OS.



Source: Pro Android 4 (Komatineni e MacLean 2012)
Figure 2.5: Detailed Android SDK Software Stack

The kernel is responsible for drivers, power management and other OS functions. At the next level are multiple well known C/C++ libraries, including: OpenGL, for 2D and 3D graphics rendering; WebKit , the most used layout engine for web pages rendering; SQLite, a widely spread relational database management system (RDBMS); FreeType, a font rasterization engine for rendering text; Apache HTTP; Open SSL; Zlib; SAX; libc; among others. Most of the time, these libraries are accessed through the Dalvik Virtual Machine (Dalvik VM), the gateway to the Android platform (Komatineni e MacLean 2012).

The Dalvik VM is a Java VM that is very optimized for space, performance and battery life. It combines the Java class files into one or more Dalvik executable files (.dex) that can be linked or run separately. Dalvik is optimized to run multiple instances of virtual

machines, and each application on Android runs on its own VM and is sandboxed in terms of resources.

Another level above is the Android Java SDK, which provides higher level libraries to access the device's resources. Most of the applications for Android are written in Java using this SDK. For applications that need to run native code in C or C++, there is the Android NDK, which is not covered here.

The Android SDK provides an abstraction for most of the resources available on the device, and is updated with new and improved packages at each new version of the system.

2.4.4 Essential Application Components

Android applications rely on some basic components from the Android API to interact with the user and the system. The two main application components are activities and services. They both can be started from another component of the app, or from the system. A brief description about each of them follows:

- *Activities* - An activity represents a screen, and is designed to perform a particular task. An application is composed of multiple independent activities, that interact between them and call each other creating the work flow of the app.

- *Services* - A service is a component that runs in the background to perform some long-running operation or work to remote processes. A service can be started and run until some other component stops it, or be launched to execute a task and stop when the task is done.

Technology	Technique	Physical / Symbolic	Absolute / Relative	LLC	Accuracy	Scale	Recognition	Limitation
GPS	Triangulation	Physical	Absolute	✓	1-5m	24 satellites	✗	Not indoors, heavy battery consumption
RFID	Proximity	Symbolic	Relative	✗	<1m	One sensor per location	✓	Must know sensor location
NFC	Proximity	Symbolic	Relative	✗	<5cm	One sensor per location	✓	Must know sensor location
Wi-Fi	Proximity / Triangulation	Physical	Absolute	✗	<200m	Known access poins: over 10.6M in Brazil, over 19.3M in USA[location-api.com]	✓	Poor accuracy
Cell ID	Proximity / Triangulation	Physical	Absolute	✗	200m - 1000m	Known cell towers: over 600K in Brazil, over 1.6M in USA[location-api.com]	✓	Poor accuracy

Table 2.2: Evaluation of the location systems using the taxonomy proposed in (Hightower e Borriello 2001).

Operating System	1Q13 Shipment Volume	1Q13 Market Share	1Q12 Shipment Volume	1Q12 Market Share	Year over Year Change
Android	162.1	75.00%	90.3	59.10%	79.50%
iOS	37.4	17.30%	35.1	23.00%	6.60%
Windows Phone	7	3.20%	3	2.00%	133.30%
BlackBerry OS	6.3	2.90%	9.7	6.40%	-35.10%
Linux	2.1	1.00%	3.6	2.40%	-41.70%
Symbian	1.2	0.60%	10.4	6.80%	-88.50%
Others	0.1	0.00%	0.6	0.40%	-83.30%
Total	216.2	100.00%	152.7	100.00%	41.60%

Source: IDC Worldwide Quarterly Mobile Phone Tracker, May 2013 (IDC 2013)
Table 2.3: Top Five Smartphone Operating Systems, Shipments, and Market Share, 1Q 2013 (Units in Millions)

Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.4%
2.2	Froyo	8	3.1%
2.3.3 - 2.3.7	Gingerbread	10	34.1%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	23.3%
4.1.x	Jelly Bean	16	32.3%
4.2.x	Jelly Bean	17	5.6%

Table 2.4: Android platform versions

3 RELATED WORK

This chapter will present some works that were researched in order to inspire and evaluate options for the development of our system. Section 3.1 presents studies that use location techniques to create context-aware computing, or that extend these techniques and could be integrated into our work as localization client applications. Section 3.2 presents Android applications on the market that automate actions in a way that is similar to the Android client application developed.

3.1 Localization

We have researched some studies that use simple location technologies to create context-aware systems. The Percontrol (Leithardt et al. 2012) is a system that manages class attendance through the use of ubiquitous technologies. It works by setting up a wireless network to which the students devices are automatically connected when they enter the classroom. Users are previously registered on the system with their devices, which allow the system to identify their presence by keeping track of when a device connects and disconnects from the network.

Other studies propose ways to extend the existent location technology. The following studies present ideas that aim to improve the precision of existent location systems. They could be integrated into our system, improving its overall performance.

The use of a robot equipped with a RFID reader is the proposal of (Hori et al. 2008) to improve the precision of RFID systems. While in the traditional use of RFID the reader is supplied with constant power, the idea of this study is to vary the reader's power to detect the distance of the tag from the antenna. First, the maximum power is set to sense tags in a long-range. Then the power is reduced in steps, which also reduces the sense range of the reader to known bounds. That allows the system to estimate the position of the tags in the environment.

The development of an indoor localization system for Android mobile devices is presented in (Sevcík 2013). Relying on the sensors of the devices, the solution merges three localization techniques to improve accuracy: received signal strength fingerprint, dead reckoning and Sequential Monte Carlo filtering. It displays the user's location on a floor map, and other layers can be added to show points of interest.

Another solution that provides Android devices with the capability of indoor localization is (Laoudias et al. 2013). The objective is to bring location based services to the indoor level, in environments such as shopping malls, museums, exhibition centres and airports. Since GPS is not a good solution inside buildings, the Airplace system uses the received signal strength data from the surrounding WiFi infrastructure to locate the user. It takes advantage of the WiFi system that is usually already deployed in the environment

and creates a "radiomap" of the place, that is then used by the Android application to locate the user indoors through state-of-the-art algorithms.

3.2 Android Automation

The idea of automating tasks on Android devices is not new. Browsing the Google Play store (Google 2013), we can find a number of applications that promise to automate actions on the occurrence of some event. The basic idea behind these applications is to allow the user to set up rules that define actions to be executed on the occurrence of a trigger event. By doing so, users can program their devices to automatically act on situations where they would execute a task manually.

Some examples of rules, that illustrate the objective of these applications, are:

- Set phone to silent mode at some specific time interval every day, like from midnight to 7 a.m.
- Turn off Wi-Fi at night to save battery, and turn it back on in the morning
- Start the music player when a headset is plugged in
- Turn on device's speaker when the phone is moved away from the ear during a phone call
- Start the navigation app when connected to the car via Bluetooth

These applications generally offer a numerous set of events that can be used as triggers. Among them are: a date or time of the day; the state of some hardware of software functionality, such as the Wi-Fi, Bluetooth, battery level and battery state (charging, discharging); the location of the device; a received call or SMS; etc.

The actions can also be chosen from an extensive set, which includes: displaying some kind of alert; changing the state of some functionality, such as mobile data, Wi-Fi, or Bluetooth; launch an application; get location; set volume; etc.

Below are listed some of the most popular apps that share these features, together with a brief description of their peculiarities.

3.2.1 AutomateIt

On AutomateIt (AutomateIt 2013), rules consist of trigger-action pairs. On the "My Rules" screen, the user can see all of his rules, activate, deactivate, remove or add new ones with the help of a wizard. The app features a "Rules Market", where the community of users can share their rules, letting others freely download and use rules directly or adapt them to their needs.

The full list of supported triggers and actions can be found on their website. As of May 2013, AutomateIt had over 100,000 downloads of its free version on Google Play. The paid version costs $2.00 and features exclusive triggers and actions. It has had over 10,000 downloads.

3.2.2 Tasker

Similarly to AutomateIt, Tasker (Tasker 2013) performs tasks based on contexts. A task is a set of actions. Actions are operations performed over one functionality of the

device. The app is announced to have more than 190 built-in actions available in 14 categories, among them are: alert, audio, media, network and phone. Tasker also allows the creation of home screen widgets that trigger tasks.

Tasker only has a paid version that, as of May 2013, costs $3.99 and has registered over 100,000 downloads on Google Play.

3.2.3 Other Applications

There are yet other apps on the market with the same purpose. Some of those, with similar functionality but less popularity then the ones described above, are *Impel* (Impel 2013), *Actions* (Actions 2013), *Android Automate* (Automate 2013) and *Automagic * Automation* (Automation 2013).

3.2.4 NFC Task Launcher

NFC Task Launcher (Tagstand 2013) differs from the previously presented apps in the way the actions are triggered. On NFC Task Launcher, instead of picking events that will trigger the execution of a task, the tasks are executed when the device reads a NFC tag.

The app offers an interface for the user to pick actions and write them on a writable NFC tag. From then on, every time this tag is read the actions stored on it will be executed. This allows users to create tags with a set of tasks and stick them to places where they would want to apply that settings. This way, reading a tag can set the phone's profile for house or work, for example.

NFC Task Launcher is the most downloaded Android application for automation of tasks found on Google Play. It only has a free version with over 500,000 downloads as of May 2013.

3.2.5 Comparison with this work

Like the applications presented in this section, the Android application we developed on this work automates the execution of tasks on the device in which it is installed. The difference begins with the fact that the existing applications respond to local triggers, while ours is a client that responds to messages from the server indicating what it should do.

Instead of using the state of the device and events that happen on it, our application executes actions based on the relationship between the owner of the device and the environment in which he is currently located.

The actions are managed by the server, which knows about the location of the user and relies on rules defined by the administrators.

Table 3.1 compares characteristics of the researched apps and this work.

	AutomateIt, Tasker and similar	NFC Task Launcher	This work
Where the actions are defined	Locally	NFC Tag	Server
What triggers an action	Events on the device	Reading of NFC Tag	Presence detected in environment
Use on multiple devices	Actions must be created for each device individually	Any device can read the tag and execute the actions	New devices can be registered to the system and actions will be assigned to them according to rules existing for their owners

Table 3.1: Comparison between existing automation applications for Android and the prototype we developed

4 MODEL

Inspired by the existent works on the field, and aiming to create an expansible system, we modeled the system as presented in this chapter. Section 4.1 will describe the terminology that we adopted. Then, in Section 4.2, we present an overview of the system components and the interactions between them. The next sections will explain individually each of the components that will work together to create the interactions in the ubiquitous environment: the server in Section 4.3; the client application types and their roles in Section 4.4; and the central database in Section 4.5.

4.1 Terminology

To facilitate the understanding of the ideas developed in this work, a terminology was created. It is presented next, and each term will be discussed along this chapter:

- **User**: Any person registered on the *system*, that will be tracked by the *localization clients* and interact with the *environments*.
- **Environment**: A specific location, of any magnitude (a campus of a university, a shopping mall, a building or a room) that will interact with the *users* present in it, based on predefined *actions* relating the environment with these *users*.
- **Functionality**: A feature controlled by a device that runs an *action client application*. For example: the temperature of the *environment* or the volume of the *user's device*. Functionalities can be modified responding to messages from the *server*.
- **Action**: A setting operation over one *functionality*. Actions are created by the *server* based on the current location of the *users*, and sent to an *action client* that will execute it. Actions relate the presence or absence of *users* in *environments* with a change in the state of some *functionality*. They can also have time restrictions.
- **Main Database / Server Database**: Database modeled and implemented to support the *system*. It comprehends all the variables taken into account on the logic that the *server* uses to define the *actions* that should be taken on any given situation.
- **Server**: The application running on a server machine that manages how the system behaves. It is always available for contact from the *localization clients* that send updates on the presence of registered *users* in tracked *environments*. It is also responsible for generating the *action* messages and sending them to the appropriate *action clients* that will execute them.

- **Client / Client Application**: General term referring to an application that interacts and is subordinated to the *server*.
- **Localization Client**: Client applications that are responsible for keeping the *server* informed about which *users* are present in the tracked *environments*.
- **Action Client**: Client applications that manage one or more *functionalities*, and therefore are responsible for the execution of *actions* on them.
- **Device**: Any machine that runs an *action client* application, or is controlled by a computer that runs it. Devices have control over *functionalities*, being the physical executors of the *actions* generated by the server.
- **System**: The system as a whole, composed of *database*, *server* and *clients* working together to create the ubiquitous *environments*.

4.2 Overview

The system implements a client-server architecture that communicates over the internet. It is composed of multiple client devices and a server backed by a database. Clients and server exchange information that generates changes on features of the environment when users enter or leave it. The result is environments that automatically adapt themselves to the preferences defined for the current context.

The client applications can be grouped into two types, according to their contribution to the system. The first group embraces the *Localization Clients*. They use one or more location technologies to determine the presence of users in covered environments. The *Localization Clients* can be part of an environment, and detect when users are within its bounds, or be associated with a user and detect in which environment the user is at a given moment.

The second group of client applications is composed of *Action Clients*. *Action Clients* are those that manage one or more functionalities related to the environment. They are accountable for changing the state of these functionalities upon receiving action messages from the server.

The server is the core of the system. It receives location information from the *Localization Clients*, defines what actions should be taken, and sends messages to the *Action Clients* demanding the execution of those actions. The reasoning to define the actions is based on a database. The database has information about every user and environment in the system, and rules associating them. These rules are analyzed at every change of context and transformed in actions that the *Action Clients* can execute, changing aspects of the environment.

Figure 4.1 depicts how these components interact. The Android client application is included in both *Action Client* and *Localization Client* groups. It locates the device's owner using NFC and the Android Location Provider, and manages device's functionalities based on where its owner is at the moment. The web page is where the system administrators will manage the system's settings. It will allow them to register new users and environments and create the action rules, for example.

To illustrate the work flow of the system, let's use a real-world example:

The environment 'laboratory-room' is located in the environment 'university-campus'. When the user 'student' enters the 'university-campus', the Android localization client on his phone locates him using the GPS and sends a message to the server telling

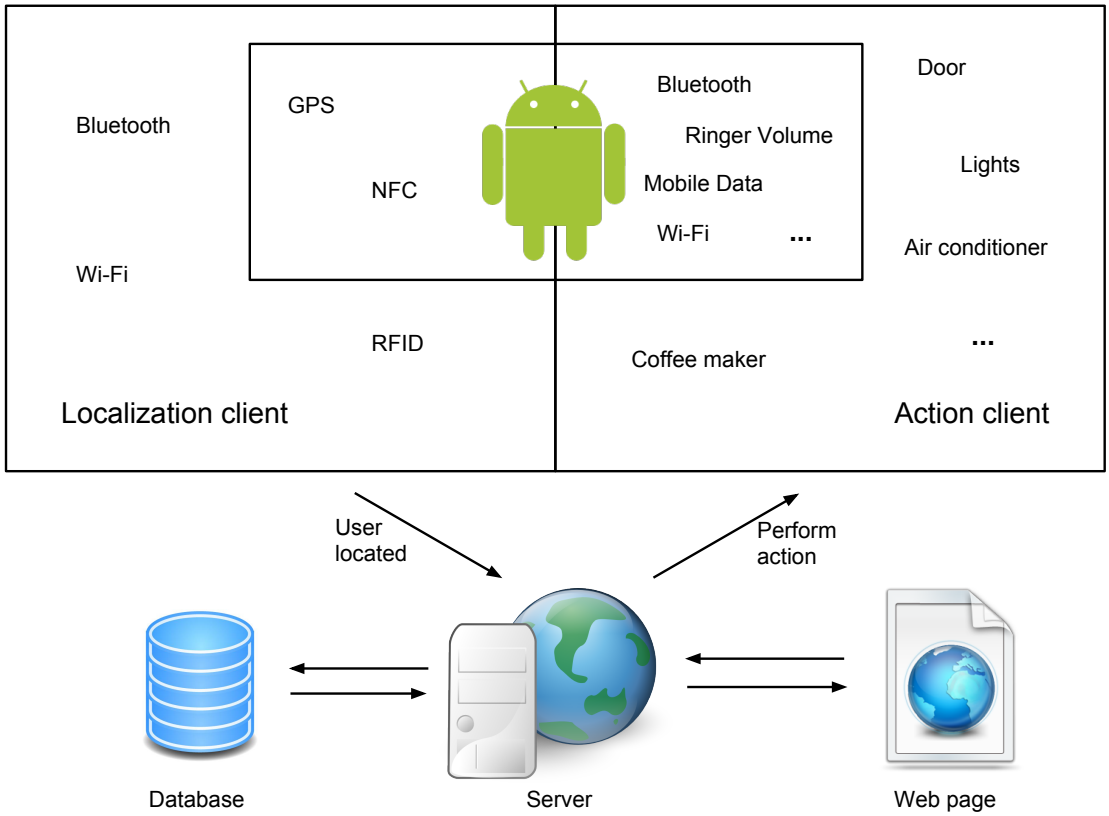


Figure 4.1: System model overview

that 'student' is located in 'university-campus'. The server processes this message and sends action messages to the action clients responsible for the execution of the actions defined for this situation. In this case, it is defined that the functionality 'Wi-Fi' of the user's device should be turned on. When the Android action client receives that message, it turns the device's Wi-Fi on.

The student then walks to the 'laboratory-room'. Since the laboratory is located indoors, where GPS reception is poor, another localization client is used to detect his presence. He touches his phone on a NFC tag at the laboratory's door. The tag holds information about the environment that the user is about to enter. The NFC action client on his phone read that information and sends a new message to the server, informing that 'student' has just entered the environment 'laboratory-room'. There are two rules defined for this situation: the volume of the student's device should be set low and the lights of the room should be turned on. The server retrieves these actions and sends them to the corresponding action clients.

When the student leaves the laboratory, he touches his phone on the NFC tag again. The action client on his device will detect that he is leaving that environment and inform the server. The server will retrieve the actions associated with the new context and send them to the action clients. Similarly, when leaving the 'university-campus', the localization client will recognize the event using the GPS and inform the server. The server will send back an action message requiring the device's Wi-Fi to be switched back to its previous state.

4.3 Server

The server is accountable for the management of the system. It keeps information about every component of the system and oversees the client applications.

Relying on the database, the server determines the current context in every environment, according to the environment and the users that are present in it. Whenever a user enters or exits an environment, the server determines how the features on this environment should respond to the new context.

Upon changes on the context, action messages are created by the server and sent to the appropriate action clients. Actions are operations that change the state of a functionality controlled by the action clients.

The server hosts a web service with a few methods that are used for communication with the client applications. These methods have three main functions:

4.3.1 Registration

New action clients need to register with the server in order to become part of the system. They need to send information that will allow the server to know when and how to contact them in the future with action messages.

To know *when* to contact an action client, the server needs to know what functionalities are supported by the client and with which user or environment the client is associated. This information is added to the database and used to define what actions will be sent to the new client.

The server also needs to know *how* it can send the actions to the client. Clients can be reached using different types of communication. For instance, if a client receives actions via a socket address, it needs to inform the server its IP address and the port number.

4.3.2 Environments query

The communication between localization clients and the server occurs only on the client-server way, therefore they do not need to register with the server. However, some localization clients that belong to users need information about the environments they should track.

Localization clients that use GPS, for example, need to locally have the coordinates for the environments that are tracked by GPS. These clients get the user's coordinates at a regular basis and check the position against the list of environments to find out if the user has just entered or exited an environment.

The server uses the *localization type* attribute of the environments to return to the client only the relevant environments. If an environment is situated indoors and detected by a NFC tag, for example, its information does not need to be sent to localization clients that use GPS.

4.3.3 Location update

To define the current context of each environment, the server needs to receive location updates from the localization clients.

When a localization client detects that a user has entered or exited an environment it sends a message to the server identifying the user, the environment, and whether the user is entering or exiting the environment. This information is processed by the server to generate the actions for the new context.

4.4 Client

Client applications are applications that assist and are subordinated to the server. They run on independent devices, and take advantage of the pervasiveness of computers and their specific functions to create a smart environment that adapts itself to the users situated in it. They exchange messages with the server to send and receive information that makes the system work.

The system was modeled with the objective of making it simple to add new client applications that just have to comply with the protocols defined. It is possible to expand the scope and improve the performance of the system by adding new clients.

Clients can be classified into two groups, according to their role on the system.

4.4.1 Localization Client

The first group of client applications is responsible for letting the server know what is happening in the environments. A localization client uses any location technology to detect when a user enters or leaves an environment, and sends that information to the server through a web service method call. Localization clients can be part of an environment or they can belong to a user.

Localization clients that belong to an environment are responsible for tracking the users that enter or leave that environment. An example is a card reader at the entrance of a room. At each card read, the client gets the identification from the user and sends it to the server together with the environment identification.

When the localization client belongs to a user, it tracks the presence of the user in the registered environments. For example, a GPS client gets the location of the user from time to time and checks if the user is entering or exiting any tracked environment.

4.4.2 Action Client

The second group of clients comprises those that change the state of features of the environment upon request from the server.

Action clients can also belong to an environment or to a user. When they belong to an environment they are static, and have control over a set of features of the environment. When they belong to a user, they run on a device that the user carries with him, and they change the state of functionalities of that device according to the context on the environment where they are.

These clients are able to receive action messages from the server, that will demand them to update the status of the functionalities they control. They are integrated into the system by registering with the server, when they inform their functionalities and how they can receive messages.

4.5 Database

The database model created for this project was designed to make the system scalable. It allows new users and environments to be registered on the fly, as well as new action and localization clients.

The simplified UML representation of the model is shown on Figure 4.2. Each table is described below:

- users: Identifies every user registered on the system and his current location, if available.

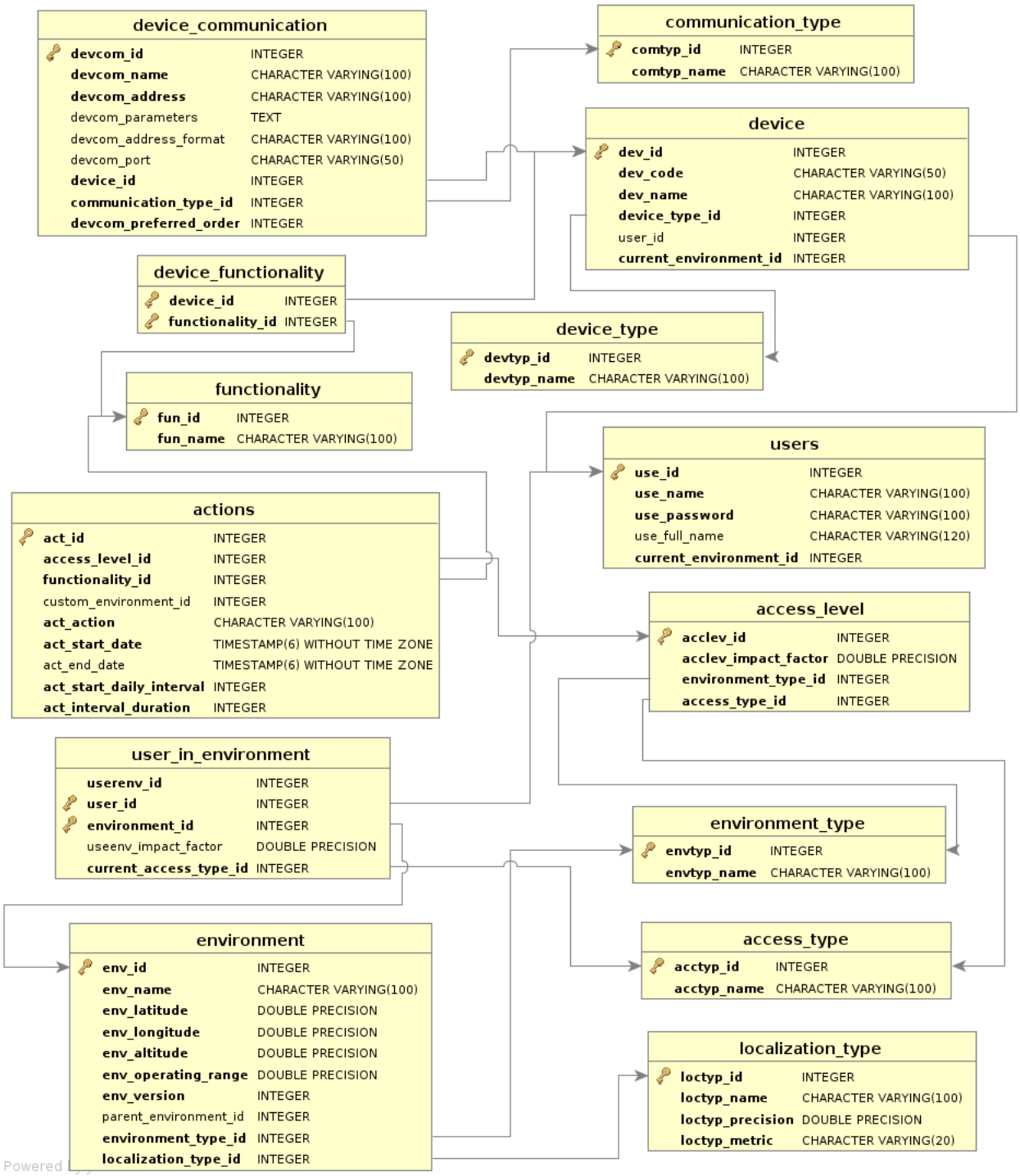


Figure 4.2: Database model for the system

- environment: Represents every environment monitored by the system. Environments are located in space by their central coordinates and a radius. When an environment is located entirely inside another one, it is said that the outer environment is the parent of the inner one.

- localization_type: Defines the method used to localize a user in the environment.

- environment_type: Enumerates the possible types of environments. The environ-

ment type defines how the environment responds to changes in the context.

- device: Stores information about every device running the action client application.
- functionality: Stores uniquely the functionalities supported by the system.
- device_functionality: Relates each device with the functionalities it has or controls.
- device_communication and communication_type: Store information about how the server can communicate with a device.
- access_type: Enumerates the possible access types that can be attributed to users. Access types group users according to their role on the environment, such as a guests or administrators.
- user_in_environment: Determines the access type that a user has in a given environment. The access type will define what actions apply to the user.
- access_level: The access level defines the impact factor for every access type in every type of environment. The impact factor is intended for use with action clients that belong to the environment, and is out of the scope of this work.
- actions: Stores the actions that should be executed on the environments at every change of context. The actions are defined based on the users present and their access type. Actions might have a start and end date, then being active only for a limited period.

5 PROTOTYPE

Based on the definitions presented in the previous chapter, we developed a prototype of the complete system in order to validate our research and modeling. We developed a server with the functionalities needed to manage the system and an application for the Android platform, which acts as both localization client and action client.

This chapter shows some details of how the prototype was implemented. Section 5.1 explains the main features of the server. Section 5.2 describes how the components of the Android client application work.

5.1 Server

The server was developed in Java using Java Servlet and the Apache Tomcat. It is a RESTful application with a few web service methods to interact with the client applications. As it receives updates on the system state, it generates and sends action messages to the registered action clients.

5.1.1 Web Service Methods

The web service methods that the server provides are called by the Android client application in order to keep the state of the system consistent. These methods use JSON to interchange data. They also use basic HTTP access authentication; that means that every request must include the authorization header.

5.1.1.1 Device Registration

This method is called by the action client at the first time it is run. It will send to the server the information needed to contact the device in the future with action messages.

The Android action client uses the Google Cloud Messaging to receive action messages. Therefore, the method for registration of such devices has the following format:

Listing 5.1: Web service method that allows the registration of new action clients

```
Method: POST
Consumes: application/json
Produces: application/json
Authorization: Basic [Base64(username:password)]
Params: deviceCode : String, communicationCode : String,
   functionalities : Array of Number, [deviceName: String].
Message format:
{
```

```
    "deviceCode":"4af28750-f250-11e2-b778-0800200c9a66",
    "communicationCode":"APA91bFk15uE15UdWu4-55pcL-
       GqQnZKlLgqwHzSx3jqhxsarueGTtcYGaoaFZQCJuospRsJ-1-OU-
       8dmgh8YHWIaW7LLVkQmDdWcpPEy4KaoxD88CcvZWXrluLaLvxp4
       Nh5Bk1rBcIBLG0t82KF_jg1u0FjXg6kvQ",
        "functionalities":[1, 3, 5, 8],
        "deviceName" : "Nexus 4 Smartphone"
}

Return: Status of the operation in the Server.
Message format return:
{       "status":"OK"    },
{       "status":"ERROR"    }, or
{       "status":"DENIED"    }
```

The *username* and *password* will be used to authenticate and associate the new device with an already registered user. The *deviceCode* is a UUID to identify the device uniquely. The *communicationCode* is the code that the device received when it registered with Google Cloud Messaging (presented in section 5.1.2). The server uses this code to reach the device when sending a message through Google Cloud Messaging. The *functionalities* is an array of identifiers of the functionalities that the device supports. The argument *deviceName* is optional, and provides a human-friendly name for the device.

The response message simply informs the result of the operation. *OK* means that the operation was successful and the device is now registered and associated with that user; *DENIED* means that the authentication failed; and *ERROR* means some error occurred during the registration. If the response is *DENIED* or *ERROR*, the device has not been registered.

5.1.1.2 Environments List

In order to track the user's presence in the environments, the Android localization client module that uses the device's sensors needs to locally have information about the environments of the system. This method provides access to elemental data about each of those environments.

Listing 5.2: Web service method that return a list of tracked environments

```
Method: GET
Consumes: application/json
Produces: application/json
Authorization: Basic [Base64(username:password)]
Query Params: currentVersion : int.
Address format: host:port/path?currentVersion=0

Return: A list of the registered environments.
Message format return:
{       "status":"OK"    },
{       "status":"ERROR"    },
{       "status":"DENIED"    }, or
{
        "status" : "UPDATED",
```

```
        "version" : 1,
        "environments" : [
                {
                        "id" : 1,
                        "version" : 1,
                        "name" : "Porto Alegre",
                        "latitude" : -30.07229614257811,
                        "longitude" : -51.17763595581054,
                        "parentEnvironment" : 0,
                        "operatingRange" : 17550.786
                },
                {
                    ...
                },
                ...
        ]
}
```

When environments are added, changed or removed from the system, the version of the environments list changes. The *currentVersion* is the version of the list of environments that the client currently has. We use it to retrieve the changes that occurred on the database from *currentVersion* to the most recent version.

The response is an update for the list of environments the client has, if needed. *OK* means that *currentVersion* is up to date; *DENIED* indicates an error in the authentication; *ERROR* indicates a general error; and *UPDATED* means the *currentVersion* is not up to date. When the response status is *UPDATED*, the server also sends the version of the most recent list of environments and the updated list itself.

5.1.1.3 Presence Detected

This method is called by the localization clients when the presence of a user in an environment is detected. The server uses it as an entry point for information about the location of the users of the system. The example below shows how the Android application uses this method.

Listing 5.3: Web service method that receives location data

```
Method: PUT
Consumes: application/json
Produces: application/json
Authorization: Basic [Base64(username:password)]
Params: deviceCode : String, environmentId : int, [exiting
   : boolean].
Message format:
{
    "deviceCode":"4af28750-f250-11e2-b778-0800200c9a66",
    "environmentId":1
    "exiting": true
}

Return: Status of the operation in the Server.
```

```
Message format return:
{       "status":"OK"     },
{       "status":"ERROR"     }, or
{       "status":"DENIED"     }
```

The *deviceCode* is used to identify the client that sent the message. The *environmentId* identifies the environment where the user was located. The optional parameter *exiting* indicates whether the user is entering or exiting the environment. When omitted, *exiting* is considered *false*.

The response is the result of the operation. *OK* means that the location was successfully processed; *DENIED* indicates an error in the authentication; and *ERROR* indicates a general error.

5.1.2 Action Messages

Action messages are the means by which the server demands actions to be executed by action clients.

The server keeps in the database the necessary information to contact each of the action clients registered on the system. When there is a change in the current state of one environment (a user enters or leaves), the server fetches the actions associated with that event and sends them to the action clients responsible for their execution.

For the Android client application, we send action messages via Google Cloud Messaging for Android (GCM) (Google 2012). Google Cloud Messaging is a service provided by Google that enhances the power of Android applications that make use of the Cloud. The service allows developers to easily send push messages with up to 4kb of payload data from the server to their applications on a mobile device. These messages are sent to Google's servers, which handle the delivery to the device. The format of the message is defined in section 5.2.2;

Push messages allow applications to save precious battery power on the communication. When the server can asynchronously contact the client, it is not necessary to have a service running on the client side just to fetch messages. The client also usually receives the messages with reduced delay.

For action clients that belong to the user, the action messages also contain the identification number of the environment. This is necessary because when the user leaves the environment, the functionalities that changed should return to their previous state. For example, if the user entered environment number 3 where his vibration alert should be turned on, an action client that belongs to him will receive a message requesting that, together with the identification of the environment he has just entered (number 3). Later, when he leaves this environment, the server sends a message informing that the user has left the environment number 3. This will make the action client return the functionalities to their previous state.

5.2 Android Client

We have chosen the Android system to be the platform used for the development of the client application prototype. Because of its openness and its dominance on the market, Android has a big community of developers and a wide documentation. The Android client application implements both *localization client* and *action client*. It was developed using the Android API 17, with backwards compatibility for the versions 2.3.3

(Gingerbread) and up. This makes the application compatible with more than 95% of the currently active Android devices (see section 2.4.2).

Due to the different levels of accuracy needed on the location of users, two *localization clients* were developed and work simultaneously on the application. The first uses the Android location services, which relies on the sensors of the device to get geographical locations. The positions gathered by this means can determine the location of the user with limited precision. They can tell us, for example, that the user is in the surroundings of a building. However, they are very poor indoors, and are not precise enough to determine in which room the user is. To locate the user more precisely in smaller environments, the second *localization client* was implemented, using NFC. With NFC it is possible to get a precise symbolic position, given that it is the NFC tag that holds information about the environment where it is placed.

The Android client application also implements an *action client* that changes the state of some basic functionalities of the device in response of commands from the server.

5.2.1 Localization Client

The localization module of the app is responsible for detecting when the device enters or leaves a known environment, and for sending that information to the server.

5.2.1.1 Localization using Android location providers

The first module of the localization client uses the device's sensors to get the user's geographical position. In general, smartphones can get location data from GPS, wireless network information and cell tower ids. Android classifies these location providers into two groups: GPS provider and network providers (which embodies wireless network information and cell tower ids).

In order to track the user's location over time, our application creates a location service that periodically request location updates from the system. The constant use of the location services, however, demands a lot of energy.

To reduce battery consumption, we use a simple algorithm to determine the location provider and time interval between position requests. This algorithm calculates the minimum distance between the user's current position and the borders of every environment. The request interval is then defined according to this distance. The closer the user is from the border of an environment, the shorter the time interval between requests will be. The location provider (GPS or network providers) is also chosen based on this distance. As GPS positions are more precise and consume more battery power, we only request them when the user is near the border of an environment.

The information about the environments that should be detected by this localization client is obtained from the server and stored in a local database. At every new location for the user, the location service tests the position against the bounds of each environment. If the user has just entered or exited an environment, the information is sent to the server.

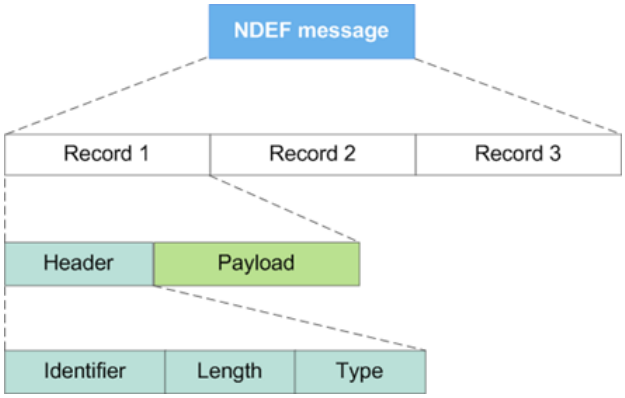
5.2.1.2 Localization using NFC

The second module implemented to work as localization client uses Near Field Communication. Android added support to NFC in 2011, on its API level 9 (Gingerbread). Among other uses, it allows the communication between the device and passive NFC tags that can store a small amount of data.

In this prototype, we use NFC as a location mean for indoor environments, and for en-

vironments where the precision of the Android location providers is not enough accurate.

The format defined by the NFC Forum for data exchange using NFC is the NDEF (NFC Data Exchange Format) message (Forum 2013). According to its specification, an NDEF message is composed of one or more NDEF records. Each NDEF record contains information about the structure of its data in a header, and a payload with the actual data. Figure 5.1 shows the general view of a NDEF message.



Source: http://www.nfc-forum.org/specs/
Figure 5.1: NDEF message structure

To represent an environment, we defined the NDEF record type *application/br-nunes.tg.environment*, where the payload identifies the environment. Then, we declared that our application handles this NDEF record type, so that the Android system will call our application whenever an NFC record with this data type is read.

To locate the user in an environment, the environment number is written to an NFC tag with our data format, and attached to the entrance of the environment. When the user touches his Android device to the tag, our application is called and has access to the payload data.

In order to identify whether the user is entering or exiting, the client holds a list of environments that the user has entered and not exited. The first time an environment number is read we assume the user is entering that environment, and add it to the list. When the tag is read again, the application detects that the user is already in that environment, and assumes he is leaving.

5.2.2 Action Client

The action client module of the Android application acts changing the state of some functionalities of the device.

On the first run, the app registers to the Google Cloud Messaging, and gets a registration identifier that is required to send messages to the device. It then registers with our server, informing this identifier and other information that will allow it to be reached later on.

5.2.2.1 Functionalities

A functionality is a feature of the Android device that can have its state changed.

In order to define what functionalities the action client would support, we researched on the Android documentation to find functionalities that the operating system allows to be programmatically changed by applications, and also used examples from the automa-

tion applications we researched (Section 3.2). We then created the following list, with the set of functionalities that are relevant in the scenario proposed by this work:

- Bluetooth
- Wi-Fi
- Mobile network data access
- Silent mode
- Vibration alert
- Ringer volume

These features are supported by the prototype application. That means that action messages from the server can modify their state or value.

5.2.2.2 Actions

The action commands are asynchronously sent from the server to the device via Google Cloud Messaging. The message informs the action client about which environment generated the actions it holds, and contains a set of key-value pairs representing the actions.

To receive the messages, the Android application declares a service class that implements some callback methods. One of this methods is *onReceiveMessage(Message message)*, that is called when a message is received. From the object *message* is possible to get the key-value pairs sent from the server, parse them and apply the desired configuration.

The key of each pair is the identifier of the functionality, and the value is the command in the JSON format:

Listing 5.4: Example of action

```
key: BLUETOOTH_STATE,
value:
{
    "state": "ON"
    "duration": 300
}
```

The duration of an action is optional. It represents the duration time of the action, in seconds. When the duration is present, the action client executes the action and restores the previous state after passed the duration time.

Table 5.1 lists the functionalities supported, their identifier keys and the format of action they support. Except for the ringer volume, all functionalities can be turned on or off. The ringer volume receives a numeric value from 0 to 100, that indicates the new volume.

Actions are executed on the device to conform to the context of the environment that the user has just entered. Therefore, when the user leaves that environment, the functionalities changed should return to their previous state. To allow that, before changing the state of any functionality, the action client saves its previous state, associating the change with the environment that caused it.

When the user leaves an environment, the server sends a different message that just informs that the user has left the environment. The action client then recovers the state of the functionalities prior to entering the environment, and restore them.

Functionality	Identifier key	JSON example
Bluetooth adapter	BLUETOOTH_STATE	{"state":"ON"}
Wi-Fi adapter	WIFI_STATE	{"state":"OFF"}
Mobile network data access	MNDA_STATE	{"state":"ON"}
Silent mode	SILENT_MODE_STATE	{"state":"OFF"}
Vibration alert	VIBRATION_STATE	{"state":"ON"}
Ringer volume	RINGER_VOLUME_VALUE	{"state":50}

Table 5.1: Functionalities supported by the Android action client module

6 VALIDATION

In order to test and validate the prototype that we developed, we created a use scenario that simulates a real use of our system. This chapter explains how this scenario was created and how the tests were executed. Section 6.1 describes the information that we added to the database to enable the use of the system. Section 6.2 outlines the workflow of the Android application, along the explanation of how the tests were performed.

6.1 Scenario

The scenario we created consists of a few users, environments and action rules relating them. The environments are shown on Figure 6.1. "Porto Alegre" is the city where all the other environments are located. "Campus Vale UFRGS" is the campus of our university. "Building 72 Informatics" is the building where "Classroom 112" and "Laboratory 205" are situated. For our tests, the environments can be of the type "Public" or "Private".

id	Name	Latitude	Longitude	Parent id	Range (m)	Localization Type	Environment Type
1	'Porto Alegre'	-30.072296	-51.177636	-	17,550.786	GPS	Public
2	'Campus Vale UFRGS'	-30.071927	-51.120080	1	903.524	GPS	Public
3	'Building 72 Informatics'	-30.068497	-51.120476	2	33.178	GPS	Private
4	'Classroom 112'	-30.068473	-51.120385	3	5.000	NFC	Private
5	'Laboratory 205'	-30.068615	-51.120616	3	5.000	NFC	Private

Figure 6.1: Environments created for validation

Then we registered some users, and defined the access type that each one has on each of the environments. The access types that we created classify the users in one of five categories for each environment: "Blocked", "Guest", "Basic", "Advanced" or "Administrator". Actions are defined relating the access types with the type of the environment. Figure 6.2 shows the values created for this scenario. According to them, we can for instance infer that the user "brnunes" has "Basic" access in the environment 5 ("Laboratory 205"). Therefore, since "Laboratory 205" is a "Private" environment, when he enters the environment his phone will have its Bluetooth, Wi-Fi and mobile data access turned ON, and the ringer volume set to 50%.

id	Username	Password	Name	Access Type in Environment				
				1	2	3	4	5
1	'borges'	12345	Guilherme Borges'	2	3	3	3	3
2	'valderi'	12345	Valderi Leithardt'	2	3	3	4	4
3	'brnunes'	12345	'Bruno Nunes'	2	3	3	2	3
4	'geyer'	12345	Cláudio Geyer'	2	3	3	4	5

Access Type	
id	Name
1	'Blocked'
2	'Guest'
3	'Basic'
4	'Advanced'
5	'Administrator'

Environment Type	Access Type	Actions: Bluetooth	Silent Mode	Vibrate Alert	Wi-Fi	Mobile Data	Volume
Private	1	OFF	ON	ON	OFF	OFF	
Private	2	ON	ON	ON	ON	ON	
Private	3	ON			ON	ON	50
Private	4				ON	ON	
Private	5				ON		
Public	2						
Public	3				ON		

Figure 6.2: Users, access types and actions created for validation

6.2 Application Workflow

With the necessary data set up in the database, we were able to start running the tests on real Android devices.

The first time the application is run on a device, the user is requested to log in with the account he has in our system. It is also possible to provide an optional name for the device. Figure 6.3 shows the screen that prompts the user for this information. When the "Login" button is touched, the device registers with the Google servers and gets the identifier that will allow it to be reached via GCM later. Then, it sends the login data and this identifier to our server, that associates the new device with the user.

As soon as the device is registered, we get to the home screen of the app. The home screen consists of a map view that shows the current position of the user and the bounds of the environments that are located using GPS. In Figure 6.4(a), we can see the environments as the circles. The blue labels were added to the screenshot to identify the environments.

The user's location is only shown if the location service is enabled. Figure 6.4(b) shows how to enable the location service. Once enabled, the app starts getting the location of the user, and detecting when he enters or leaves any environment located by GPS. In our scenario, the location service instantly detects that we have entered the environments with id 1, 2 and 3. Figure 6.4(c) shows the moment when it is detected that the user entered the environment with id 1 ("Porto Alegre").

Soon after the localization client sends the location data to the server, we receive the action messages, and the action client executes them. In our case, since the Wi-Fi and

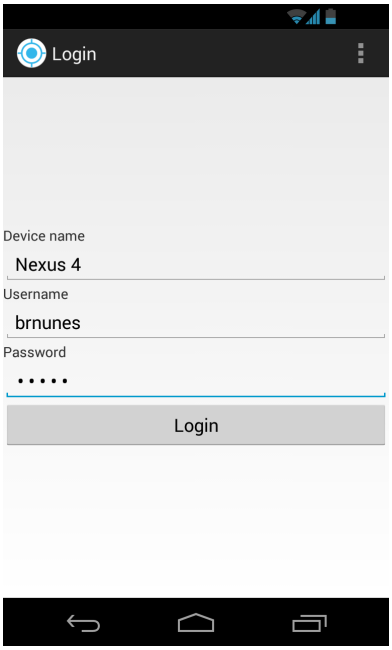


Figure 6.3: Application login screen

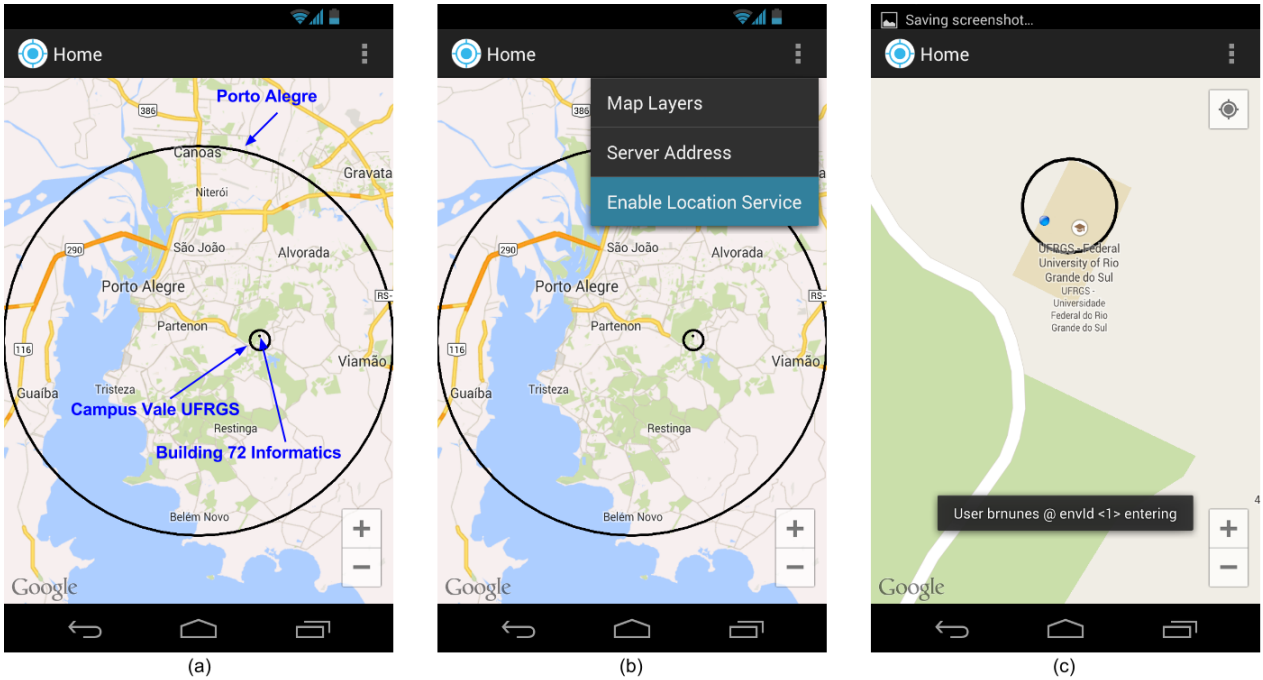


Figure 6.4: Application home screen

mobile data access were already on, the action client turns on the Bluetooth and sets the device's volume to 50%. Whenever an action is executed, a notification sound is played (if the device is not in silent mode) to inform the user of the change.

In order to test the localization client that uses NFC we used some NFC tags. For each of the two environments that are located using NFC we wrote their identifier number in an NFC tag, and placed the tag at the entry of the environment. Using this location method, the presence of the user in the environment is detected when the user reads the tag with his smartphone, as shown in Figure 6.5.

Figure 6.5: Device reading NFC tag

When an NFC tag written with the data format that we defined is read, the Android system delivers its content to our application, even if the application is not running on the foreground. From the payload data, the application gets the environment identifier and sends to the server, that generates the appropriate action messages.

The event of the user leaving the environment is also sent to the server. The server will use it to define the new context on the environment. It will also inform the occurrence of the event to the action clients that belong to the user, allowing them to undo the changes done when the user entered this environment.

6.3 Results

Our tests simulating a real use of the system have showed satisfactory results. The localization clients were precise when locating the user in the tracked environments, detecting when the user moved in and out the environments. However, when the the user was placed for a long time exactly on the border of an environment located by GPS, the application eventually detected that he entered and exited the environment multiple times. This happened because of the accuracy of the localization methods used. The accuracy of GPS, Wi-Fi and Cell IDs positioning system is not better than a few meters, what causes the application to get different locations even when the user is still. One solution for this problem would be to use a "transition area", a few meters around the borders of the environments, where the positions are not considered.

The action client properly performed the actions sent by the server. As expected, the application stored the current state of the functionalities before changing them. This allowed it to correctly set the functionality back to its previous state when we left an environment that had generated actions.

7 CONCLUSION

This work was born from the realization of the pervasiveness of computers in our daily life. During the course of a day, a normal person interacts with multiple computers, many of them having specific purposes, controlling aspects of the environment that we usually want to change to match our preference. We wanted to create a system that knows the user's preferences, and perform changes to the environments proactively based on them.

The system was modeled using a client-server approach, so that we could have a fully working system developed as a prototype, and it could be expanded by future works with little or no changes to its core.

Our prototype was developed to automatically apply settings to mobile devices, based on the context. Smartphones have become the Swiss army knife of modern life: one single device that embeds enough technology to be people's all-purpose computer. We took advantage of this power, and created an application that turns them into agents of our system. They generate the data that we need in order to perform context-aware computing, and at the same time they are the clients that execute actions as an automatic response to the change of context.

During the course of the development of this project we faced and overcame diverse challenges. The need for localization methods with better accuracy to track smaller sized environments was resolved by creating a symbolic localization method, using NFC, which proved to be very effective. To obtain a better response time to actions, we made use of push messages, that brought also the benefit of reduced battery consumption.

At the end, we were able to model a system that can be expanded and manage the diverse features of environments that can be controlled by computers. We also could build a fully functional system that validated the model proposed, by using the currently available technology.

7.1 Future Work

Our biggest goal with this work was to create a system that could be easily expanded. We have accomplished that by creating a model that simplify the integration of new clients. Future works could use the protocols we defined for the communication between server and clients, and increase the reach of the system. The expansion can by achieved through the integration of new action clients that control different features of the environments, or localization clients that use different location methods.

In the Android application, some new features could be added to improve the user experience. One of these features is a screen to manage the app settings, where the user can select which functionalities can be changed by the system and which can not. This would give more power to the user, allowing him to restrict our application from changing the

state of functionalities that he wants to have full control (the Wi-Fi adapter, for example).

The battery consumption of the device is another issue that could be addressed by future works. Although we used an algorithm for getting the user's location only when needed, therefore avoiding waste, the battery consumption has naturally increased with the constant use of the application when the location service is on. For a new version of the application, one could implement a new algorithm or use another technologies that allow the application to request less positions from the Android location providers.

REFERENCES

[Actions 2013]ACTIONS. *Actions*. 2013. Visited May 2013. Available from Internet: <https://play.google.com/store/apps/details?id=com.sigmacel.actionsworksfree>.

[Alliance 2007]ALLIANCE, O. H. *Alliance FAQ*. 2007. Visited July 2013. Available from Internet: <http://www.openhandsetalliance.com/oha_faq.html>.

[Alliance 2007]ALLIANCE, O. H. *Android Overview*. 2007. Visited July 2013. Available from Internet: <http://www.openhandsetalliance.com/android_overview.html>.

[Alliance 2013]ALLIANCE, W.-F. *Wi-Fi Alliance*. 2013. Visited July 2013. Available from Internet: <http://www.wi-fi.org/>.

[Arduino 2013]ARDUINO. *Arduino*. 2013. Visited August 2013. Available from Internet: <http://www.arduino.cc/>.

[Ashton 2009]ASHTON, K. *That 'Internet of Things' Thing*. 2009. Visited August 2013. Available from Internet: <http://www.rfidjournal.com/articles/view?4986>.

[Automate 2013]AUTOMATE, A. *Android Automate*. 2013. Visited May 2013. Available from Internet: <https://play.google.com/store/apps/details?id=com.monkeysoft.automate>.

[AutomateIt 2013]AUTOMATEIT. *AutomateIt - Make your smartphone smarter!* 2013. Visited May 2013. Available from Internet: <http://automateitapp.com/>.

[Automation 2013]AUTOMATION, A. *Automagic Automation*. 2013. Visited May 2013. Available from Internet: <http://automagic4android.com/en/>.

[Center 2013]CENTER, U. C. G. N. *Global Positioning System*. 2013. Visited June 2013. Available from Internet: <http://www.navcen.uscg.gov/?pageName=gpsFaq>.

[Forum 2004]FORUM, N. *NFC Forum: Nokia, Philips And Sony Establish The Near Field Communication (NFC) Forum*. 2004. Visited July 2013. Available from Internet: <http://www.nfc-forum.org/news/pr/view?item_key=d8968a33b4812e2509e5b74247d1366dc8ef91d8>.

[Forum 2013]FORUM, N. *Simple NDEF Exchange Protocol - Technical Specification*. 2013. Visited September 2013. Available from Internet: <http://www.nfc-forum.org/specs>.

[Google 2012]GOOGLE. *Google Cloud Messaging for Android*. 2012. Visited May 2013. Available from Internet: <http://developer.android.com/google/gcm/index.html>.

[Google 2013]GOOGLE. *Google Play*. 2013. Visited May 2013. Available from Internet: <https://play.google.com/store/apps>.

[Hightower e Borriello 2001]HIGHTOWER, J.; BORRIELLO, G. *Location Sensing Techniques*. [S.l.], 2001.

[Hightower e Borriello 2001]HIGHTOWER, J.; BORRIELLO, G. Location systems for ubiquitous computing. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 34, n. 8, p. 57–66, ago. 2001. ISSN 0018-9162. Available from Internet: <http://dx.doi.org/10.1109/2.940014>.

[Hori et al. 2008]HORI, T. et al. A multi-sensing-range method for position estimation of passive rfid tags. In: *Proceedings of the 2008 IEEE International Conference on Wireless & Mobile Computing, Networking & Communication*. Washington, DC, USA: IEEE Computer Society, 2008. (WIMOB '08), p. 208–213. ISBN 978-0-7695-3393-3. Available from Internet: <http://dx.doi.org/10.1109/WiMob.2008.39>.

[IDC 2013]IDC. *IDC - Press Release*. 2013. Visited August 2013. Available from Internet: <http://www.idc.com/getdoc.jsp?containerId=prUS24108913>.

[Impel 2013]IMPEL. *Impel*. 2013. Visited May 2013. Available from Internet: <http://impel.phase2i.com/>.

[INFO 2013]INFO, E. R. *What is RFID?* 2013. Visited July 2013. Available from Internet: <http://www.epc-rfid.info/rfid>.

[Juniper 2011]JUNIPER. *Press Release: 1 in 5 Smartphones will have NFC by 2014, Spurred by Recent Breakthroughs: New Juniper Research Report*. 2011. Visited June 2013. Available from Internet: <http://www.juniperresearch.com/viewpressrelease.php?id=308&pr=239>.

[Komatineni e MacLean 2012]KOMATINENI, S.; MACLEAN, D. *Pro Android 4*. 1st. ed. Berkely, CA, USA: Apress, 2012. ISBN 1430239301, 9781430239307.

[Laoudias et al. 2013]LAOUDIAS, C. et al. Airplace: Indoor geolocation on smartphones through wifi fingerprinting. *ERCIM News*, v. 2013, n. 93, 2013.

[Leithardt et al. 2012]LEITHARDT, V. et al. Percontrol: A pervasive system for educational environments. In: *International Workshop on Mobility and Communication for Cooperation and Coordination, IEEE International Conference on Computing, Networking and Communications (ICNC 2012)*. [S.l.: s.n.], 2012.

[Milette e Stroud 2012]MILETTE, G.; STROUD, A. *Professional Android Sensor Programming*. Wiley, 2012. (ITPro collection). ISBN 9781118240458. Available from Internet: <http://books.google.com.br/books?id=dZjo-254FucC>.

[Navigation e Timing 2013]NAVIGATION, N. C. O. for S.-B. P.; TIMING. *Global Positioning System*. 2013. Visited June 2013. Available from Internet: <http://www.gps.gov/>.

[Schilit et al. 1994]SCHILIT, B. et al. Context-aware computing applications. In: *Proceedings of the 1994 First Workshop on Mobile Computing Systems*

and Applications. Washington, DC, USA: IEEE Computer Society, 1994. (WMCSA '94), p. 85–90. ISBN 978-0-7695-3451-0. Available from Internet: <http://dx.doi.org/10.1109/WMCSA.1994.16>.

[Schwarz 2011]SCHWARZ, R. . *Near Field Communication (NFC) Technology and Measurements - White Paper*. 2011. Available from Internet: <http://www.rohde-schwarz.com/en/applications/near-field-communication-nfc-technology-and-measurements-application-note_56280-15836.html>.

[Sevcík 2013]SEVCÍK, J. Indoor user localization using mobile devices. *ERCIM News*, v. 2013, n. 93, 2013.

[Tagstand 2013]TAGSTAND. *NFC Task Launcher*. 2013. Visited May 2013. Available from Internet: <http://launcher.tagstand.com/>.

[Tasker 2013]TASKER. *Tasker - Total Automation for Android*. 2013. Visited May 2013. Available from Internet: <http://tasker.dinglisch.net/>.

[Weiser 1991]WEISER, M. The computer for the 21st century. *Scientific American*, v. 265, n. 3, p. 66–75, set. 1991.

[Weiser 1993]WEISER, M. Some computer science issues in ubiquitous computing. *Commun. ACM*, ACM, New York, NY, USA, v. 36, n. 7, p. 75–84, jul. 1993. ISSN 0001-0782. Available from Internet: <http://doi.acm.org/10.1145/159544.159617>.

[Yang et al. 2013]YANG, P. et al. Efficient object localization using sparsely distributed passive rfid tags. *IEEE Transactions on Industrial Electronics*, v. 60, n. 12, p. 5914–5924, 2013.