Pedro Emanuel Albuquerque e Baptista dos Santos

Licenciado em Engenharia Informática

Personalization Platform for Multimodal Ubiquitous Computing Applications

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Nuno Manuel Robalo Correia, Prof. Catedrático,
Universidade Nova de Lisboa

Júri:

Presidente:	Prof. Dr. José Augusto Legatheaux Martins
Arguente:	Profª. Drª. Ana Paula Pereira Afonso
Vogal:	Prof. Dr. Nuno Manuel Robalo Correia

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Dezembro, 2013

Personalization Platform for Multimodal Ubiquitous Computing Applications

À minha mãe

Acknowledgements

I would like to thank my advisor, Prof. Nuno Correia for all of the good advises and guidance during the development of this dissertation. Another very important person was my former teacher and PhD student Rui Neves Madeira, with whom I worked closely and greatly contributed to the success of this work. I would also like to thank all the students and researchers at the CITI's IMG group for all of the suggestions and help testing out some ideas and prototypes.

Finally, I thank my family for all the support during this important stage of my personal and academic life. I am especially grateful for all the love and comprehension of my mother who has been always besides me when I need her. Without her this dissertation would surely not have been possible.

Abstract

We currently live surrounded by a myriad of computing devices running multiple applications. In general, the user experience on each of those scenarios is not adapted to each user's specific needs, without personalization and integration across scenarios. Moreover, developers usually do not have the right tools to handle that in a standard and generic way. As such, a personalization platform may provide those tools.

This kind of platform should be readily available to be used by any developer. Therefore, it must be developed to be available over the Internet. With the advances in IT infrastructure, it is now possible to develop reliable and scalable services running on abstract and virtualized platforms. Those are some of the advantages of cloud computing, which offers a model of utility computing where customers are able to dynamically allocate the resources they need and are charged accordingly.

This work focuses on the creation of a cloud-based personalization platform built on a previously developed generic user modeling framework. It provides user profiling and context-awareness tools to third-party developers.

A public display-based application was also developed. It provides useful information to students, teachers and others in a university campus as they are detected by Bluetooth scanning. It uses the personalization platform as the basis to select the most relevant information in each situation, while a mobile application was developed to be used as an input mechanism. A user study was conducted to assess the usefulness of the application and to validate some design choices. The results were mostly positive.

Keywords: Personalization Platform, Application Personalization, Web Services, Multimodal Interaction, Mobile and Ubiquitous Computing.

Resumo

Atualmente vivemos rodeados por uma diversidade de dispositivos de computação, em múltiplos cenários de interação. Geralmente, a experiência de utilização em cada um desses cenários não está adaptada às necessidades de cada utilizador, não existindo personalização e integração entre cenários. Além disso, os programadores não possuem as ferramentas adequadas para lidar com o problema de uma forma normalizada. Como tal, uma plataforma de personalização poderá fornecer essas funcionalidades.

O uso deste tipo de plataforma deve ser facilmente acessível para qualquer programador. Como tal, esta tem que ser desenvolvida de modo a estar acessível através da Internet. Com os avanços nas infraestruturas de informação, é atualmente possível desenvolver serviços fiáveis e escaláveis sobre uma plataforma abstrata e virtualizada. Essas são algumas das vantagens da computação em nuvem, que oferecem um modelo de *utility computing* onde os clientes são capazes de alocar dinamicamente os recursos de que necessitam e que são pagos conforme a utilização.

Este trabalho visou a criação de uma plataforma na *cloud* para personalização, integrando uma *framework* genérica de modelação de utilizadores previamente desenvolvida. Fornece ferramentas de criação de perfis de utilizador e *context-awareness* a terceiros.

Uma aplicação centrada num ecrã público foi também desenvolvida. Esta fornece informação útil a alunos, professores e outros num campus universitário, sendo estes detetados numa procura por Bluetooth. Utiliza a plataforma para personalização como forma de selecionar a informação mais relevante em cada situação, sendo que uma aplicação móvel foi desenvolvida para ser utilizada como mecanismo de interação. Um estudo de utilizador foi conduzido para determinar a utilidade da aplicação e para validar algumas escolhas de desenho. Os resultados foram maioritariamente positivos.

Palavras-chave: Plataforma de Personalização, Personalização de Aplicações, Serviços Web, Interação Multimodal, Computação Móvel e Ubíqua.

Contents

List of Figures

List of Tables

Listings

1

Introduction

Personal computers, smartphones, tablets, and other computing devices, are common in the everyday life of the population of most countries. People interact with a myriad of different applications running on multiple computing devices. All of this technology is not well integrated, even with the current state of the Internet and the World Wide Web, with the social networks and the so-called web applications, coming from multiple vendors.

There is some integration in the case of web applications that are available under the same vendor umbrella, such as Google[1], Facebook[2] or Microsoft[3]. However, in most cases there is no connection between different applications. A seamless integration of user preferences and habits across every application used on a daily basis does not exist.

Besides, we are currently surrounded with so many displays of different sizes and purposes. Yet those displays are not usually designed to work together in order to offer a convergent experience across them. They are typically unaware of each other, not reacting to changes in any of the other devices and displays.

Therefore, it is important to tackle this problem by allowing applications to be more pervasive and ubiquitous. This concept of ubiquitous computing aims to unobtrusively integrate computing in people's daily activities, proactively reacting to them in order to improve their life experiences [Wei91].

[1] `https://www.google.com/intl/en-US/about`
[2] `https://www.facebook.com/facebook`
[3] `http://www.microsoft.com/about/en/us/default.aspx`

1.1 Motivation

Usually, applications do not share a common knowledge and view of the users. That does not allow developers to fine tune the user experience of the applications they create. This means that the users are not getting exactly what they want and need, and developers are unable to give them that with the current level of tools and technology. This intended human-centered vision demands adaptivity, for personalization.

It may be interesting to define personalization to better understand what it is. According to the Oxford Advanced Learner's Dictionary [Oxf13], **personalize** can refer to *personalize something: to mark something in some way to show that it belongs to a particular person* or even *to design or change something so that it is suitable for the needs of a particular person*. Even though personalize can be used with other different meanings, these two are the ones that are of interest for this work.

The goal is to make the applications aware of who is using them, creating a sense of connection between the user and the application. Also, even more important, is the ability of designing applications from the ground up that are able to change according to who is using them. This kind of objectives may be taken for granted in some industries, such as the super cars market in which buyers may personalize many aspects of the product delivered to them, or in restaurants where you can personalize a dish to your tastes. However, in the computing world, aside from changing some aesthetics of an application or the placement of some tool bars and buttons, it is not usual to have a personalized product according to the ever evolving needs of the users. Even if this goal is not trivial it should be achievable, considering that applications are much more dynamic than other physical products.

Besides personalization itself, which is something that could theoretically be achieved in a localized and application specific way, the really interesting part is leveraging the power of existing mobile and ubiquitous devices. This ubiquity has been largely possible thanks to the existence of the Internet, by interconnecting multiple users and applications. Taking that into account, it is also natural that personalization should be taken to the web, being appropriate for both third-party developers and for the personalization concept itself. Developers should be able to personalize their applications without having to implement everything from scratch.

It is the advantage and utility of having personalized applications, and the power of pervasive computing devices, that make personalization desirable. With more and more developers and users interested in personalization, it is natural that new and better approaches are developed.

Additionally, there is the desire of pushing forward ubiquitous computing through the integration and convergence of multiple devices and displays into a unified user experience. Such scenario creates a very interesting test bed for new interaction experiments with applications that run across multiple devices and displays.

1.2 Description and Context

One of the main goals of this work was the development of a cloud based system to help developers in the process of applying personalization techniques to ubiquitous computing systems and applications. Those techniques can be used at different levels including the multimodal interface of the applications and the way interaction is handled. With such objective in mind, it was developed a web services platform based on a generic personalization model that defines how personalization should be applied to a system or application.

A developer registers with the platform to get access to the services. Then, it is possible to register systems being developed in order to use the API[4] collection and make ubiquitous personalization requests. The platform integrates a user profiling algorithm module.

The platform allows developers to give their users a better experience through a service that developers can rely on to provide personalization tools and in which users can trust with their data. To reach such goals, a robust and highly available service, with mechanisms that put users in control of their personal data, was developed.

Moreover, the research investigated how the personalization model could be applied to applications running across different devices, whether they are mobile or stationary. With that in mind, there was a focus on studying applications that run across mobile devices (e.g., smartphones and tablets) and (large) situated public displays.

Therefore, to help with that study, an ubiquitous computing application that takes advantage of the service was also developed. This application is a public display-based application that leverages the capabilities of the personalization platform to give users an experience tailored for them.

This work was developed in collaboration with the IMG[5] group of CITI[6] at the Department of Informatics[7] of the Faculty of Sciences and Technology[8], Nova University of Lisbon[9]. It follows the work being done at IMG in the area of personalizing applications, in particular a PhD work of a member [Mad12]. Moreover, the personalization service is based on a generic personalization model that defines how personalization should be applied to a system or application. That model has been presented in a previous Master's dissertation [VC12], also under the scope of the more wider PhD work.

Besides developing an application from the ground up to take advantage of the personalization service, other existing applications developed at the IMG laboratory may also use it in the future.

[4] Application Programming Interface
[5] Interactive Multimedia Group - `http://img.di.fct.unl.pt`
[6] Centro de Investigação em Informática e Tecnologias da Informação - `http://citi.di.fct.unl.pt`
[7] DI/FCT/UNL - `http://www.di.fct.unl.pt`
[8] Faculdade de Ciências e Tecnologia - `http://www.fct.unl.pt`
[9] Universidade Nova de Lisboa - `http://www.unl.pt`

1.3 Solution

The final developed solution consists of a cloud based service to develop personalization-enabled applications. This service was developed to provide a personalization platform, from now on called **P^{2}MUCA**. This name comes from the title of this dissertation: *Personalization Platform for Multimodal Ubiquitous Computing Applications*.

P^{2}MUCA provides a set of services to application developers to ease the introduction of more personal features and interaction when building their applications. Developers will naturally have to study the platform's documentation to learn the available services API and configuration options. But other than that, developers have access to a set of operations and algorithms that can be applied over user data and combined according to their specific needs.

The provided functions and configuration modules allow the developer to choose between different options that can be relevant for a given application. For instance, for an *Application A* it may be interesting to get information about the age group of its users, while another *Application B* might need to know the location of its users and the associated weather conditions. Also, it is possible to combine different personalization data resources into different configurations and giving to each one different weights. The platform allows third-party developers to have some limited extensibility over the predefined features. For example, adding new context sources is possible as long as they are just symbolic context, i.e., it does not depend of an external service such as weather, so the value passed as user context is used directly in the personalization process.

P^{2}MUCA targets two types of users:

- **Developers** that will have to register their client applications to use the service;
- **End users** that will allow applications to have access to their data.

This dichotomy between developers and end users is similar to what we have in services such as Facebook and Twitter. As with those services, end users have the ability to authorize applications to use their profile data. This is extremely important because P^{2}MUCA may store privacy sensitive information. Therefore, P^{2}MUCA also focus on providing that kind of control to end users.

The development of an ubiquitous computing inspired application was also very important to validate the work being done in the personalization solution. The main goal was to create a public display-based application that leveraged the capabilities of P^{2}MUCA. Such kind of applications are usually placed in a public or semi-public environment and run on relatively large screens (i.e., 40 inches or more). Besides the public display itself, there is a major focus on expanding it by using mobile devices (e.g., smartphones and tablets) to provide input for the application. The public display and the private display of a mobile device work together to give users the best experience possible.

The main objective of the application is to provide useful information tailored to those who are passing by the public display. Additionally, it provides interactivity to those that wish to control the application. For testing purposes, the application was initially deployed at the IMG lab so that it could be tested by some teachers, students and researchers. Some of the included features are: a) information about public transports; b) current traffic information if a user drives a car; d) news titles; e) lunch menus on multiple bar and restaurants on campus; f) private messages; g) weather conditions; among other useful information personalized to each member of the academic community. Considering the theme of this application, it was *codenamed* **FCT4U**.

1.4 Main Contributions

The main contributions made by this work are the following:

- **Personalization platform (P^{2}MUCA)**: Development of a personalization services platform based on a meta-model that can describe user profiles, context and interaction data. This platform can be used by third-party developers to build personalization-enabled applications;

- **Context-aware public display application (FCT4U)**: Creation of a public display-based application that can be used to present relevant information to users, by sensing who is around in order to display information tailored to them. A companion mobile application was developed as part of FCT4U. That application is used for interaction with the public display and context sensing. It was built as a mobile application for Android smartphones and tablets.

- **User analysis**: Usability tests were performed to measure both the quality of the public display interface design and the personalization services provided by the developed platform;

- **Publications**: A full paper [AMCed] focusing on the development of FCT4U and the study of mobile-public display interaction was accepted at the UIC '13[10] conference. A short paper regarding the development of the personalization model and its deployment in P^{2}MUCA was also submitted to IUI '14[11] conference and it is awaiting review [Mad+14].

1.5 Report Structure

This document is structured in five different chapters:

[10]The 10th IEEE International Conference on Ubiquitous Intelligence and Computing - `http://cse.stfx.ca/~uic2013/`

[11]2014 International Conference on Intelligent User Interfaces - `http://www.iuiconf.org/`

- **Chapter 1 - Introduction**: This first chapter is focused on presenting a general overview of the dissertation. The main focus areas are the motivation and involving context, the description of the problems it tries to solve, an introduction about the developed solutions for those problems and what contributions can be derived from them;

- **Chapter 2 - Related Work**: In this chapter it will be discussed the research of related work and technologies in each relevant areas of this dissertation. Overall, there will be a focus on web services and cloud computing technologies, interaction with public displays and personalization platforms for applications.

- **Chapter 3 - P^{2}MUCA**: This chapter focus on the development of P^{2}MUCA, it explains how some design decisions were made, how the system was built, what features are available and how to use them.

- **Chapter 4 - FCT4U**: This is the chapter dedicated to the FCT4U development. It goes through some of the choices made during the design of the application, presents the system architecture and available features and ends by exploring the results of the user tests that were conducted.

- **Chapter 5 - Conclusions and Future Work**: The last chapter is dedicated to present the conclusions about the developed work and some ideas to further improve P^{2}MUCA and FCT4U.

2

Related Work

The main focus of this chapter is to present previously developed projects and studies in the fields that this dissertation covers. This chapter will be further divided into three sections:

- Web Services and Cloud Computing
- Personalization of Applications
- Interaction with Public Displays
- Geographical Data Analysis

2.1 Web Services and Cloud Computing

An important part of this dissertation is the development of a modern web infrastructure, capable of providing services to other applications over the Internet and being able to scale to ever growing demands. As such, the use of Web services and cloud computing technology is almost mandatory. In this section, the main concepts of those two subjects will be covered.

2.1.1 Web Services

The W3C[1] defines a "Web service" as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine processable format, specifically Web Services Description Language (WSDL).

[1]World Wide Web Consortium - `http://www.w3.org/`

Other systems interact with the Web service in a manner prescribed by its description using SOAP[2] messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." [HB04].

However, as stated by the W3C, this definition is used without prejudice to other definitions. Moreover, other similar definitions can also be considered acceptable. For example, by loosening the use of W3C Web services specific standards like WSDL and SOAP. So, in this dissertation, unless when specifically talking about W3C Web Service standards, the adopted definition is a more broad one. In fact, W3C also states that two different classes of Web services can be found [Boo+04]:

- REST-compliant Web services, in which the primary purpose of the service is to manipulate XML [Mal+04] or JSON[3] representations of Web resources using a uniform set of "stateless" operations;
- Arbitrary Web services that expose an arbitrary set of operations.

Both cases use URIs[4] to identify resources and use web protocols (e.g., HTTP and SOAP) and open data formats (e.g., XML or JSON) for messaging and communication. Web services are platform-independent and based on structured messages (XML, JSON or other formats). The idea is to distribute services over the Internet, making them available for clients. These services can be implemented or consumed with any programming language. The language/platform agnostic characteristics are specially interesting for P^{2}MUCA, which aims to be widely adopted by developers, regardless of their favorite development tools.

2.1.1.1 W3C Web Services

Web Services, which follow the W3C standards, are usually composed of three distinct entities at a high level architectural view [HB04], [Boo+04]. Those are:

- **Service provider**: This is the provider of the web service. The service provider implements the service and makes it available for consumers;
- **Service requester**: This is any consumer of the web service. The requester invokes an existing web service by opening a network connection and sending SOAP messages;
- **Service registry**: It is a centralized directory of services. The registry is used as a central place where providers or developers can publish new services and find existing ones.

A logical overview of the relationship between these entities can be seen in Figure 2.1. First, the web service provider publishes its Web services to the service registry. Next, the

[2]Simple Object Access Protocol
[3]JavaScript Object Notation - http://www.json.org/
[4]Uniform resource identifier

web service consumer looks for desired Web services by searching the registry. At last, the client invokes the Web services by using the information obtained from the registry.

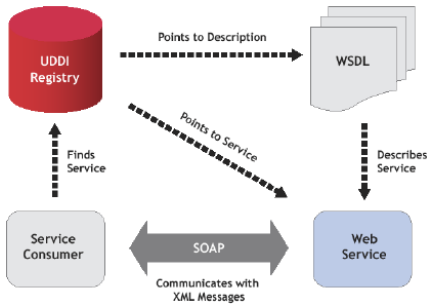


Figure 2.1: Web Services logical view

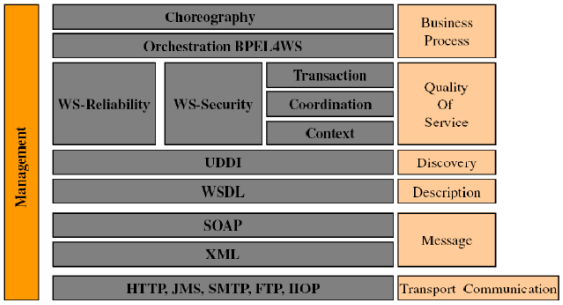


Figure 2.2: Web Services technology stack

In order to implement this architectural view there is a set of W3C standards. This set of formats and protocols can be seen as a technology stack (see Figure 2.2). That stack is composed of multiple layers: transport layer (e.g., using HTTP[5]); messaging layer (e.g., mainly using SOAP[6]), description layer (e.g., through WSDL[7]); discovery layer (e.g., using UDDI[8]).

Besides these main layers there are other important ones that can be inserted in-between. These specifications are generally referred to as WS-*. Some of those extra specifications are [Erl05]: WS-Security; WS-Reliability; WS-Transaction; WS-Adressing; WS-Policy; WS-MetadataExchange; WS-Notification.

Web Service Development

First, a programmer builds a web service with a specific programming language and the service is published using WSDL to describe its interface. This interface presents the set of operations provided by the web service.

The web service must be deployed in a server container to make it available to consumers. On the client side, an object representing remote Web services must be generated, allowing clients to call the operations defined on the server side's service interface. The developer does not have to worry about manipulating SOAP messages, because this is usually done by a library provided by the programming language. Also, as the objective of Web services is to be interoperable, a web service developed in C# on the .NET platform running on Windows can be accessed by a client written in any other language/-platform running on any other operating system. This is a quality that is very useful for P^{2}MUCA, allowing anybody to use its services regardless of the programming language and operating system they are running.

[5]Hypertext Transfer Protocol
[6]Simple Object Access Protocol
[7]Web Services Description Language
[8]Universal Description, Discovery and Integration

Web Services Characteristics

From what we have seen so far it is possible to enumerate some key determining features of Web services [Pap08], [tut13]:

- **XML-based**: Web Services rely on XML for data representation and transportation. The use of XML avoids any network, operating system or platform dependency;
- **Loose coupling**: There is no direct tie between a web service and its users, contrasting with tightly coupled systems, where the client and server logic are closely bound to each other;
- **Ability to be synchronous or asynchronous**: The interaction between consumers and the web service can be made in a synchronous or asynchronous manner. Asynchronous invocations allow clients to do a request and then immediately execute other operations without waiting for the result;
- **Supports RPC**[9]: Web services enable clients to invoke methods and operations on remote objects using SOAP. Thus basically enabling the remote invocation of any code that is made available through the web service;
- **Enables reusability**: Web services can be remotely accessed, providing a way to make a pre-existing code available through the network. As a result, that code can be used by multiple applications;
- **Interoperability**: Web services enable communication between different applications, running on different operating systems and developed in different programming languages. Thus, it enables interoperability across systems and applications;
- **Standard Formats and Protocols**: Web services use well defined industry standard protocols and formats across all the technology stack, promoting interoperability This standardization gives organizations many advantages, like a wide range of choices, reduction of costs due to competition and increase in the quality;
- **Automatic Discovery**: Automatic discovery mechanisms allow to easily find web service descriptions that have been previously published.

2.1.1.2 REST

Representational State Transfer (REST) is a style of software architecture for distributed systems such as the World Wide Web. The term was coined in 2000 by Roy Fielding in his doctoral dissertation [Fie00]. REST was later further developed at W3C Technical Architecture Group. The largest implementation of a system that conforms to the view of the REST architectural style is the World Wide Web.

[9] Remote Procedure Call

The REST-style architecture consists of clients and servers. Clients initiate requests to servers that process those requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources. A resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of a resource.

REST is less strongly typed than its counterpart, SOAP. The REST language uses nouns and verbs, and has an emphasis on readability. Unlike SOAP, REST does not require XML parsing and does not require a message header to and from a service provider. The core principles of REST are:

- **Resource identification through URI**: Resources are identified by URIs which provide a service discovery mechanism.
- **Uniform interface**: A uniform interface between clients and servers simplifies and decouples the architecture, which enables each part to evolve independently. The guiding principles of this interface are: a base URI for the web service; the use of an Internet media type for communication (e.g., XML or JSON); a set of operations supported by the web service using HTTP methods; an hypertext driven API;
- **Client-server**: Clients are separated from servers by a uniform interface. This separation of concerns means that clients are not concerned with data storage, while servers are not concerned with the user interface or user state. Servers and clients may also be replaced and developed independently, as long as the interface is not changed;
- **Self-descriptive messages**: Resources' content can be presented in several formats (e.g., XML, JSON, plain text, etc.). Metadata about the resource can be used to detect transmission errors and perform authentication or access control;
- **Stateless clients**: Client-server communication is constrained by no client context being stored on the server between requests. Each request from any client contains all of the information necessary to service the request, and any session state is held by the client;
- **Caching**: As on the World Wide Web, clients are able to cache responses. Therefore, they implicitly or explicitly define themselves as cacheable, or not, to prevent clients from reusing outdated or inappropriate data. Well-managed caching partially helps improving scalability and performance;
- **Layered system**: Clients are not usually aware whether they are connected directly to the end server or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load balancing and by providing shared caches. They may also enforce security policies;

Web services using a REST approach are usually called RESTful services. In such services, information elements are seen as resources and each one is linked to a global identifier, i.e., an URI. Clients and servers communicate through a standardized protocol (HTTP) and exchange representations of those resources. Clients must possess the resources' identifiers and understand the semantics of each HTTP method, in order to manipulate resources. Besides that, there is no need to know the internal implementation of the services and system configuration, i.e., whether there are caches, proxies, firewalls, tunnels, or anything else between the client and the server which hosts the resources. However, clients must be able to interpret the returned data format, which is often an XML or JSON document.

RESTful Services Methods

REST uses different HTTP methods to describe the operations that can be done over a resource. The four main HTTP methods are:

- **GET**: It is used to read information about a resource. It can be used to list all the members of a collection or to retrieve the information about a member of a collection;
- **POST**: This method is used to update the information about a resource;
- **PUT**: The PUT method is used to create new resources;
- **DELETE**: It is used to delete a resources.

Using these general semantics of the HTTP methods, it is possible to implement all the four basic operations over data, usually referred as CRUD (Create, Retrieve, Update, Delete). Moreover, it is possible to model any kind of desired operation.

Advantages and Disadvantages of RESTful web services

As discussed in [PZL08], RESTful has some aspects that can be seen as positive. Some of those aspects are:

- RESTful services appear to be simple because REST applies many existing well-known standards (HTTP, XML, URI, and MIME). It also only needs infrastructure that has already become common and ordinary;
- HTTP clients and servers are compatible with all programming languages and operating system/hardware platforms, and the default HTTP port 80 is usually left open by default in most firewall configurations;
- Only a small effort is needed to build a client for a RESTful service. Services can be tested using simply a mere web browser and the development of client software becomes superfluous;

- REST allows discovering web resources without any discovery or registry repository.

However, there are some downsides of using this simpler approach. For example, it is impossible to encode arbitrarily large amounts of data into the resource URI, because servers are not ready to deal with such long URIs. It may also be challenging to encode complex data structures into URIs. However, some HTTP methods (e.g., POST) do not suffer from such limitations, being able to carry data in the request body.

Unlike SOAP-based web services, which have a standard vocabulary to describe the web service interface through WSDL, Restful web services currently have no such grammar. Both the service consumer and service producer must have an out-of-band agreement. However, services can be described using WADL[10]. It is an XML-based file format that provides a machine-readable description of RESTful web services, but it is not yet widely used.

2.1.1.3 RESTful Services vs. W3C Web Services

A good comparison between RESTful services and W3C Web services (from now on called *WS-** for short) is made in [PZL08]. Table 2.1 was based on that comparison and summarizes some of the most relevant differences.

	WS-*	**RESTful services**
Transport	HTTP, FTP, SMTP, etc.	HTTP
Messages	SOAP	HTTP requests with custom data
Loose coupling	Yes	Yes (even more loosely coupled)
Contract design	Code-first or Contract-first	Contract-less
Message exchange	Request-response and One-way	Request-response
Interface Description	WSDL	WADL
Service Discovery	UDDI	No formal definition
Service Composition	BPEL[11]	Mashups of disparate services
Extra features	Set of WS-* standards	Ad-hoc or using extensions of related technologies (e.g., HTTPS for security)

Table 2.1: WS-* vs RESTful services

It is important to note that WS-* and REST services can be implemented and consumed in any programming language or platform, as long as needed libraries and tools are available.

[10]Web Application Description Language

2.1.2 Authentication and Authorization

Authentication and authorization are important concepts for P^{2}MUCA. The first thing that needs to be well understood is that those are two distinct concepts. Whereas authentication is the process of verifying that *you are who you say you are*, authorization is the process of verifying that *you are permitted to do what you are trying to do*. Therefore, authorization usually depends upon authentication to check whether someone is permitted to do something.

These concepts are well within the domain of security and there are many different ways to cover such requirements. The most usual way is through the use of a username and password for authentication, and once an identity is established then authorization is checked by looking up some sort of access control list to verify if someone is allowed to do something before proceeding.

Since P^{2}MUCA deals with personal and private information it is very important that an authorization mechanism is put in place. Some sort of authentication also needs to be in place to support such authorization mechanism.

This would be easy on a single application using a classic username and password and a set of permission rules. However, for a system like P^{2}MUCA, a major use case for the authorization system is the ability of third-party applications performing actions on behalf of a given user. The obvious solution would be to provide the username and password of the user to a third-party application and use them to authenticate with the system. However, this is often undesirable because it would give a third-party application full access to a user account. That full access could be exploited by a malicious third-party developer or by someone who hacked the third-party application. Therefore, a system in which a user gives only some limited permissions, which can be revoked at any time, is important.

There are standards such as OpenID [spe07] to solve the authentication part of the problem. It allows user to be authenticated using a third-party services called identity providers. Users can choose to use their preferred OpenID providers to log in to applications that accept the OpenID authentication scheme.

The authentication follows roughly these steps:

1. The user gives the application his OpenID URL;
2. If the user is not authenticated, it redirects her/him to the corresponding OpenID provider login screen;
3. The user is authenticated with the OpenID provider;
4. The OpenID provider tells the application that the authentication was successful;

This solves the problem of sharing user authentication across different applications. However, without further extending the protocol, it does not address the authorization part of the problem.

There is another standard called OAuth [OAu13] to deal with authorization. It was designed to allow users to share their private resources on one application with other applications without having to hand out their credentials (i.e., username and password). Instead, authorization tokens are issued to authorized applications. Those tokens may have limited access rights and time validity, and can be revoked by users if an application misbehaves. This approach puts users in control of their data.

Currently, OAuth is widely used by Facebook, Twitter and Google APIs, among many others. The most current version of OAuth is the 2.0 version [Int12b], but many services still rely on OAuth 1.0 [Int12a]. The newer version is currently a *Proposed Standard* at IETF[12]. It is built to be simpler and easier to use and implement, also removing some of the shortcomings of OAuth 1.0, such as better support for non-web applications.

OAuth 1.0 still has the advantage when it comes to ensuring authentication and integrity of message at the cost of being much more complex to implement. Messages are digitally signed using a key and a hash function (HMAC[13]). But this mechanisms does not ensure message confidentiality, which must be ensured by the transport layer, i.e, using HTTPS[14]. So, to make OAuth 2.0 simpler, all cryptographic protections were dropped and instead the specification relies solely on HTTPS to provide authentication, integrity and confidentiality.

The developer of a third-party application must first register it with the service before using OAuth. When registering a new application, basic information such as application name, website, logo, etc., must be filled. A redirect URI is also defined to return users back to the application.

Each registered application has a client ID and a client secret. The client ID is considered public information, and is used to build login URLs while the client secret must be kept confidential. A token grant flow must be followed to get a token that allows the application to act on behalf of a user:

1. An authorization request is submitted to the service server, which validates the client;

2. The server redirects the user to a page requesting access to its resources;

3. If the user authorizes the third-party application to access its data, the server redirects the user back to the application along with an authorization code;

4. The application makes an out-of-band request to the server and exchanges the authorization code for an access token.

So at this point the server has an access token which is a username/password equivalent for the user. It can make requests to the content provider on behalf of the user by passing that access token as part of the request.

[12]Internet Engineering Task Force
[13]Hash-based message authentication code
[14]Hypertext Transfer Protocol Secure

However, that token is usually not permanent. If that is the case then an additional refresh token is issued. It is important that such token is never transmitted over a insecure channel to avoid eavesdropping. The idea is that even if the access token is compromised it will only work for a limited time (e.g., an hour). This way the damage can be limited, while the refresh token can be used to get new valid access token.

Thus, OAuth can be used to get a token that allows a third-party application to make request to another application or service on behalf of a user. This corresponds to what is needed for the personalization platform. Also, even though OAuth is not built with authentication in mind, it is possible to extend it easily to allow such scenario. For example, by making a request with the token it is possible to make a simple API call on a service that returns the user identity, which can be used to establish authentication.

Besides OpenID and OAuth, there are many more technologies that deal with these issues. In fact, as already stated for W3C Web services, there is WS-Security and other related standards that deal with many security issues, including authentication and authorization (e.g., WS-Trust and WS-Federation).

There is also a more general standard called SAML[15] [SAM13]. It is an XML-based framework for exchange security and identity information across applications. SAML deals with assertions about subjects. Those represent statements about authentication, attributes, authorization, etc. It has multiple profiles that support a variety of use cases, but it can be further extended if needed, and integrates well with W3C Web services technology. However, for simple REST/HTTP based services it adds a lot of complexity when compared with OpenID and OAuth.

2.1.3 Cloud Computing

Considering the current trends, it is only natural that any component developed in the scope of this dissertation, which is created to be remotely accessible, namely the P^{2}MUCA, should be built with cloud computing in mind.

2.1.3.1 Cloud Definition

First of all, it is important to define what cloud computing is. According to [Hur+09], the cloud is a business and economical model, and most definitions agree that providing a paid utility service is a cornerstone of cloud computing. The provided utility services can be of various nature, from virtual infrastructure to a computing platform, or even end user software products. These services are usually delivered in an on-demand and pay-as-you-go basis, creating the concept of *utility computing*.

Definitions by multiple authors are explored in [Vaq+08], while trying to reach a consensual and encompassing one. There are authors, such as Klems, Kaplan and Edwards, who consider that the cloud must be rapidly scalable and easily adaptable *on-demand*. Other authors, such as Gourlay, Sheynkman, Harting and Berger also consider that the

[15] Security Assertion Markup Language

cloud is in its essence a kind of resource virtualization. Sultan even suggests that it should be automated. This makes perfect sense because the only way to get on-demand allocation of resources and a fast scalability is through the (usually automated) virtualization of resources that can be dynamically allocated.

All these concepts are somewhat related to grid computing. So, there are authors (e.g., Doerksen and Buyya) that consider cloud computing as a relative of grid computing. In fact, the two are related and cloud computing uses some of the same concepts and technology that was developed to support the vision of grid computing. However, it follows a more pragmatic approach that so far has been more successful than grid computing. Experts disagree on how really cloud and grid computing are actually interrelated and try as much as possible to disambiguate the two concepts. Moreover, there are those (e.g., Half and Kepes) who simply define the cloud based on the services it provides. For example, IaaS (Infrastructure as a Service), PaaS (Platform as a Service) e SaaS (Software as a Service).

Finally, considering the *NIST* definition of cloud computing [MG11], [ZCB10] definition of cloud computing, a possible definition that encompasses most visions is:

> "Cloud computing is a business model in which service providers offer physical and logical resources, virtualized or not, with a high degree of scalability according to costumer's needs (on-demand) and billed according to the used resources (pay-as-you-go). All of this is done in a quick and possibly automatic way, providing computational resources as a service (utility computing)."

2.1.3.2 Types of Cloud Computing

Cloud computing can be categorized in multiple types, depending on whether the access to the resources is open or restricted to a given organization. However, it is important to note that cloud computing it is still in the first phase of Gartner's hype cycle (called *Positive Hype*) [Vaq+08] and so it is still growing and evolving. As such, the definitions presented below may become obsolete as the cloud continues to evolve in the next years.

Public Cloud

In this type of cloud the services and infrastructure are provided by external organizations that rent access to resources. As such, those resources are shared between multiple customers and are dynamically allocated and deallocated according to the customers' needs. A simple view of a public cloud can be seen in Figure 2.3(a).

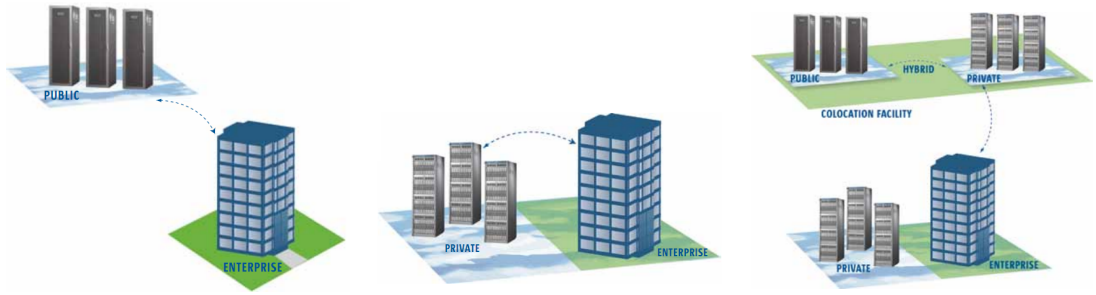
Private Cloud

A private cloud is designed to be used exclusively within an organization (see Figure 2.3(b)). The private infrastructure might be maintained by the organization that will use it or that can be outsourced to another company that will provide exclusive access to the

contracted infrastructure. This type of cloud is usually chosen when more control and security are required.

Hybrid Cloud

An hybrid cloud combines public and private clouds (see Figure 2.3(c)). In this scenario, part of the infrastructures is private and another part uses resources in a public cloud. In general, it provides more flexibility than public and private clouds because both can be combined for the desired results. It offers better control and security than a public cloud, while at the same time it provides more *on-demand* scalability when compared to private clouds.



(a) Public cloud (b) Public cloud (c) Hybrid cloud

Figure 2.3: Types of cloud computing

Intercloud

The intercloud term is mentioned in a Cisco white paper *Cisco Cloud Computing* [Cis09]. According to their vision, the intercloud will eventually emerge as a public, open, and decoupled cloud-computing internetwork. In a sense, the intercloud would be an enhancement and extension of the Internet itself. The intercloud will decouple resource consumers from cloud resource providers, allowing to find resources on demand with providers. FraSCAti [MRS11], [OW213] is a research project in this area which was able to run multiple applications across thirteen cloud providers by using a middleware to deploy SCA[16] applications.

2.1.3.3 Cloud Architecture and Business Models

The cloud architecture can be divided, at a high abstraction level, into four layers. Figure 2.4 shows an overview.

On top of this overall architecture, cloud computing follows a business model in which services are offered according to customers' needs, i.e., the providers lease their hardware and other logical resources as a service that follows an on-demand policy. The hardware and infrastructure layers give origin to IaaS (Infrastructure as a Service), the

[16]Service Component Architecture - It is a model for composing applications according to SOA (Software Oriented Architecture) principles: `http://www.oasis-opencsa.org/`

platform layer to PaaS (Platform as a Service) and the application layer to SaaS (Software as a Service). There are more specific cases that may cross boundaries between layers and these definitions, but meanwhile it is important to define each of these more general cases [Hur+09], [ZCB10].

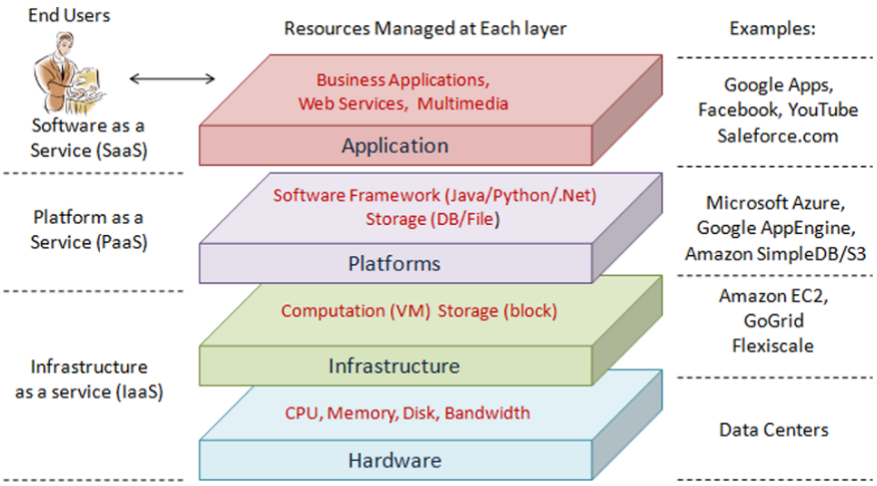


Figure 2.4: Cloud architecture [ZCB10]

IaaS (Hardware Layer and Infrastructure Layer)

The hardware layer comprises all of the physical resources of the cloud. It comprises servers, routers, switches, cooling and power distribution. There must be special attention to hardware configuration and maintenance, fault tolerance and resource management.

The infrastructure layer emerges on top of the hardware layer. It offers a resource pool of computational and storage resources. Virtualization technology is used to abstract, encapsulate and partition the underlying physical resources.

From a business perspective, the infrastructure layers gives origin to IaaS. It represents the idea of renting virtual hardware resources as needed. This way, organizations do not have to buy and maintain hardware. Keeping up with the latest technology becomes a problem of the provider instead of the customer [Rim+11].

The most known service of this kind is quite possible the Amazon Elastic Compute Cloud[17]. But there are many other public and private IaaS products, e.g., Rackspace, GoGrid, VMWare, HP, OpenStack and Eucalyptus.

PaaS (Platform Layer)

The platform layer is placed on top of the infrastructure layer, consisting of: operating systems, application frameworks and software environments. The business model for

[17] Amazon EC2 - `http://aws.amazon.com/ec2/`

this layer is called PaaS and aims to give developers a platform that includes all the tools that are needed from the beginning to end of the development cycle. The underlying hardware is virtualized and isolated between customers, with no direct control over it.

PaaS can be very useful as a way to build applications with a set of well known tools that can be easily deployed in a scalable and reliable environment. As such, this is the kind of service that will be used to deploy P^{2}MUCA.

Some widely known PaaS products are Google App Engine, Windows Azure, Heroku by Salesforce, OpenShift by Red Hat and Cloud Foundy by VMWare. Those platforms provide support for programming languages, databases, data storage, sandboxed runtime environment, etc., to be used by applications.

SaaS (Application Layer)

We find the application layer at the top of the hierarchy. This layer usually offers ready to use applications that can be accessed through a web browser, as such it follows the SaaS model.

These applications, unlike more traditional desktop applications, can take advantage of the automatic scaling and elasticity that the underlying layers provide. This way, it is possible to reach a higher level of performance, scalability and availability with lower operational costs. The software is offered through an on-demand policy over the Internet to multiple tenants.

Customers should take into account the scalability, security, performance and fault tolerance in a shared environment. They should also consider the ease of configuration and integration with the rest of the organization's IT infrastructure.

Examples of commercial SaaS applications include Google Apps, Facebook, YouTube, Salesforce.com and Office 365.

2.1.3.4 Platform as a Service

Of all the cloud service types the most relevant for P^{2}MUCA is **PaaS**. While IaaS could be used to deploy virtual machines to run the developed software, directly managing those virtual machines instances would involve creating some kind of management layer that would deviate from the main objectives of this dissertation and would be a major development overhead. On the other hand, SaaS is something that is at a too high level, since it offers specific services and software.

PaaS offers an environment that serves as a platform to build and deploy applications and services. It can be used to develop and run P^{2}MUCA. Therefore, it is important to look at some PaaS products.

Google App Engine

Google App Engine (GAE) [Goo13c] is a PaaS product by Google. It was first launched to the public as a preview in September 2008, and has moved out of preview in September 2011. It allows its users and customers to develop web applications and run them in

Google's public cloud platform. Currently, it supports developing applications in Java, Python and Go. GAE is run and managed by Google, spawning across multiple data centers, and it has an SLA[18] of 99.95% uptime per month [Goo13e]. There is also professional support for Premier accounts [Goo13d].

Applications run on a sandbox environment in Google's servers. The sandbox isolates applications in its own secure, reliable environment that is independent of the hardware, operating system and physical location of the server. It also provides isolation between different applications ensuring security and performance in a multi-tenant environment. The use of a small isolated environment allows GAE to distribute web requests for the application across multiple servers for scalability. However, this environment imposes some limitations that must be taken into account when evaluating the possibility of using this platform [Goo13g] [Blo10].

GAE is built on top of Google's own private cloud [ZZL12]. Even if that cloud is private and proprietary, there are some known components. For example, Google has its own distributed file system called Google File System (GFS), a distributed storage system for structured data called BigTable, which works as a sparse, distributed and persistent multi-dimensional sorted map to store arbitrary data [Cha+08] [Cha+06].

In terms of data storage, GAE has multiple options:

- **App Engine Datastore**: A non-relational object datastore. It provides a query engine and atomic transactions;
- **Google Cloud SQL [Goo13f]**: A service providing a MySQL-compatible database.
- **Google Cloud Storage**: A data service that provides a storage service for really big files and objects.
- **Memcache**: Used to cache temporary data in memory for faster access.

Other services available are e-mail sending, image manipulation, authentication and authorization based on Google Accounts and a scheduling service to run tasks at a predefined time and interval, among other options [Goo13g].

GAE is used for commercial purposes but also on research projects (e.g., [Blo10]). The fact that Google provides free quotas[19] that allow development and testing, a good set of tools and documentation are important factors when choosing GAE. In fact, a 2010 paper about the need of including web development as a subject of undergraduate Computer Science courses, has suggested the use of GAE as a learning platform [HP10].

However, GAE is not perfect. One of its major problems is that it promotes vendor lock-in. This is due to the fact that GAE provides many specific APIs and services. So, if an application uses them, it becomes unportable to another platform without modifications. There are some open source implementations that offer Google App Engine

[18]Service Level Agreement

[19]Google App Engine Quotas - `https://developers.google.com/appengine/docs/quotas`

compatibility (e.g., AppScale[20] and TyphoonAE[21]), but they are not fully compatible with GAE. So, if independence from a single vendor is important, GAE may not be the best choice

OpenShift

OpenShift[22] is a PaaS product by Red Hat. It is available as a public cloud service, as a private cloud solution (OpenShift Enterprise[23]) and is also available as an open source project (OpenShift Origin[24]). It was initially launched on May 4th, 2011[25].

OpenShift uses unaltered open source language runtimes and frameworks. It does not have proprietary technologies. This ensures application portability to and from OpenShift thus, avoiding vendor lock-in. The public service offers free access to limited resources that can be used for development.

It supports multiple programming languages and frameworks, namely Node.js, Ruby, Python, PHP, Perl, and Java. Besides those, the platform can run binary programs as long as they are also able to run on Red Had Enterprise Linux (the operating system used by OpenShift). This allows customers to use almost any language and framework.

As other PaaS products, OpenShift takes care of managing the underlying infrastructure and scaling applications as needed. It provides disk space, CPU resources, memory, network connectivity, and an Apache or JBoss server. It enables customers to create, deploy and manage applications in this cloud platform. The basic functional units of the platform are the following:

- **Broker**: It is the single point of contact for all application management activities. It is responsible for managing user logins, DNS, application state, and general orchestration of the applications.
- **Cartridges**: Provide the actual functionality necessary to run the user application. There are multiple cartridges for multiple programming languages, frameworks and application servers (e.g, PHP, Ruby on Rails, JBoss, etc.). There are also database cartridges for PostgreSQL, MySQL, MongoDB, etc..
- **Gears**: Provide a resource-constrained container to run one or more cartridges. Multiple *gears* run on a single physical or virtual machine node. Gears are generally over-allocated on nodes since not all applications are active at the same time, which helps getting the most out of the available physical hardware.
- **Nodes**: Host the cartridges that are made available to users and the gears where user applications are actually stored and run. Each node is responsible for receiving

[20] http://appscale.cs.ucsb.edu/index.html
[21] http://code.google.com/p/typhoonae/
[22] https://openshift.redhat.com/
[23] http://www.redhat.com/products/cloud-computing/openshift-enterprise/
[24] https://github.com/openshift
[25] https://openshift.redhat.com/community/blogs/announcing-openshift-the-platform-as-a-service-for-developers-who-love-open-source-and-cdi

and performing the actions requested by the broker.

Application scaling provides automatic allocation of resources based on demand. OpenShift monitors the resource requirements of scaled applications, and increases or decreases resources accordingly. An overview of the components and architecture seen so far is schematized in Figure 2.5.

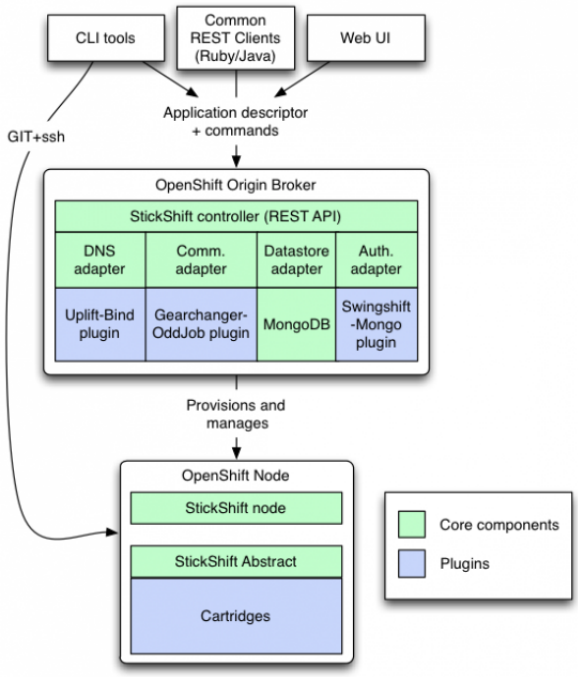


Figure 2.5: OpenShift's architecture overview

Besides being an open-source project that can be provided by third-party companies or used within an organization, OpenShift offers more control and flexibility than any other evaluated PaaS product by allowing developers to customize and build their own cartridges, going as far as to offer direct SSH access to gears.

This comes with the price of some extra complexity when deploying applications. However, for the most common application types there are templates that are relatively easy to use. For example, a simple Java EE application may be deployed simply by cloning a Git repository, copying a Maven[26] project or WAR[27] file into a predefined location, committing the changes and pushing them to the OpenShift gear so that it can be deployed. After that the application is made available at the configured URL. Also, even if some extra work is needed, unsupported types of applications may also be deployed on OpenShift due to its configurability.

Cloud Foundry

Cloud Foundry is another PaaS solution, launched by VMware in April 12th, 2011[28].

[26] Apache Maven - `http://maven.apache.org/`

[27] **W**eb application **AR**chive)

[28] `http://www.vmware.com/company/news/releases/cloud-foundry-apr2011.html`

Since then, a new division of VMware has been created called GoPivotal[29] and has taken over the maintenance of CloudFoundry.

CloudFoundry.com is the public instance operated by GoPivotal. It supports Java, Ruby, and Node.js and it offers PostgreSQL, RabbitMQ, MongoDB, MySQL and Redis services. These platforms and services match must be supported by vendors that wish to be considered Cloud Foundry compatible.

After being freely available for a long time as a public beta, the public service was released as a commercial product in June 2013 without a free offer. However, there are also third-party providers commercializing services based on the open source code made available by GoPivotal[30] that have free plans. This open source approach avoids vendor lock-in and promotes interoperability and competition between different PaaS providers.

A key part of CloudFoundry is VCAP[31] which is composed of five main components and a messaging bus [Zyg11]:

- **Cloud Controller**: It is responsible of managing the whole platform;
- **Health Manager**: It periodically checks the application state and if a problem is detected it tells the Cloud Controller to correct it;
- **DEA (Droplet Execution Agents)**: An agent that is run on each node that executes applications;
- **Router**: This component is responsible for routing requests to the running application instances;
- **Services**: Services consist of two parts: the service itself (e.g., MySQL) and a management layer that establishes connections between applications and the services;
- **NATS**: A simple publish-subscribe messaging bus used for communication by other components.

An important design choice when designing Cloud Foundry is to make it as independent as possible from the underlying infrastructure. It is possible to run it on top of VMWare vSphere, Amazon EC2 and OpenStack, etc. This is one more way of avoiding vendor lock-in and since it is open-source, an organization may also possible to deploy it into a private cloud on-premises.

2.2 Personalization of Applications

The main objective of this dissertation is the development of P^{2}MUCA, a personalization platform that can be used by third-party developers to enrich the experience of their

[29] GoPivotal, Inc. - `http://www.gopivotal.com/`
[30] `http://core.cloudfoundry.org/listings`
[31] VMWare Cloud Application Platform

applications. As such, it is important to find what kind of solutions have already been developed. Most projects in this area focus mainly on recommending information to users. Thus, they are usually called recommender systems and may be used to power recommendation on e-commerce web sites such as Amazon[32] or eBay[33].

2.2.1 Social Itinerary Recommendation from User-generated Digital Trails

Planning a travel to unfamiliar regions is a difficult task for novice travelers. To help them, researchers propose a social itinerary recommendation system by learning from multiple user-generated digital trails (e.g., GPS trajectories of residents and travel experts) [Yoo+12].

This is another example of a recommendation system. Even though there is not much focus on automatic personalization, the system has a few preferences that can be set to help suggesting the itinerary that is more interesting and relevant to the user.

Data was gathered from local residents to validate their approach, by using GPS devices to record their itineraries. They also set up a series of user tests and a testing platform. The deployment was made following a Mobile/Cloud architecture. In the mobile part, they tried to keep the number of tasks to a minimum and only included processes that are light and that cannot be processed elsewhere. The cloud part is where the system calculates the itinerary recommendations and executes data mining routines. This is similar to what P^{2}MUCA does. Figure 2.6 shows the architecture implemented for the system.

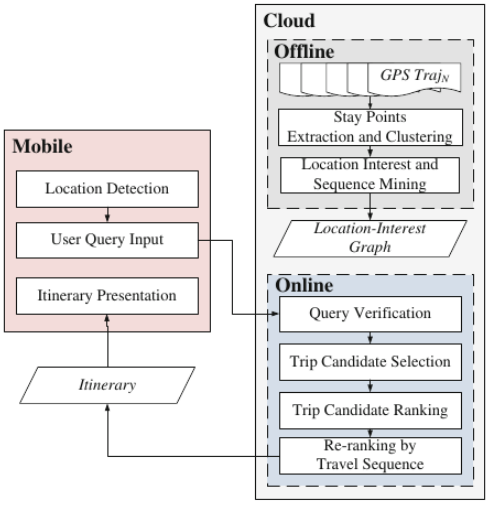


Figure 2.6: Itinerary recommender architecture [Yoo+12]

The researchers also made another version of the test platform that run exclusively on mobile phones. As expected, the performance was much worse when processing a

[32] http://www.amazon.com/
[33] http://www.ebay.com/

considerable amount of data. However, it was faster to use the local processing power for smaller workloads, thus avoiding communication latency.

2.2.2 My Own Web

An interesting view over personalization is given by My Own Web [LY05]. They rightly identified that as the Web continues to grow in size, the problem of information overload while browsing and searching becomes more severe. Personalization might be the key to alleviate this problem, by customizing the web towards the user personal interests and preferences. This personalization should present users' preferred information at the right moment. In order to do that, the system has to collect and analyze relevant information and store the results in the users' profile.

The concept of Web Personalization can be defined as the process of customizing the contents and structure of a Web site to the specific and individual needs of each user based upon the user's behavior [LY05]. In order to achieve it, the main steps are the collection of web data, the preprocessing of that data, the analysis of the collected data and finally the decision making and final recommendation.

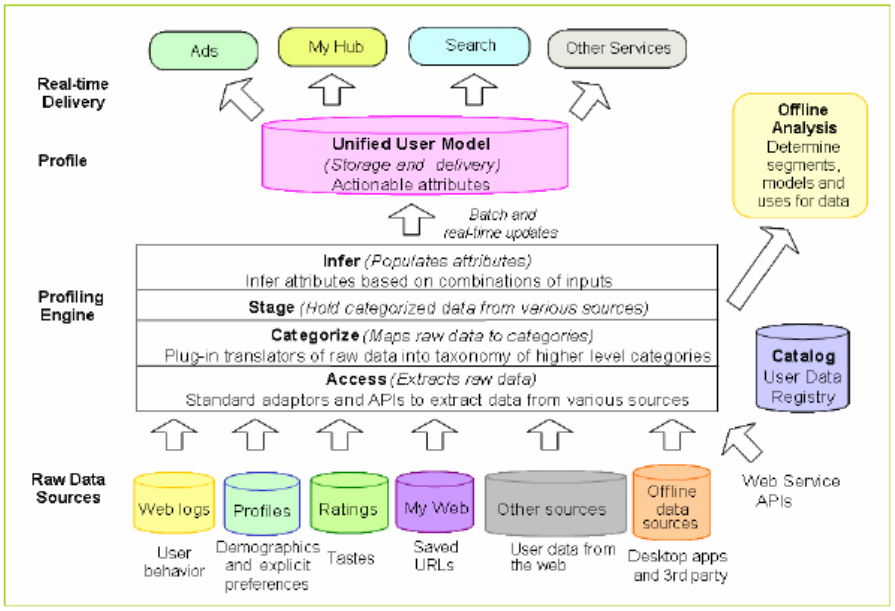


Figure 2.7: Overview of the Unified Personalization Platform [LY05]

Authors proposed an Unified Personalization Platform, which can be envisioned as a raw data collector of user profiles, web logs and other sources from the Web or PC to be fed into the Profiling Engine. The engine extracts raw data, maps them into categories, holds the categorized data, and then populates attributes into appropriate services that need them to enable personalization. A graphical overview of this process can be seen in Figure 2.7.

According to the conceptual design of *My Own Web*, a Personalization Platform seamlessly integrates users' preferences, information and media across the Web. Their vision

is that it should not simply be restricted to personalization on a specific website. It should allow users to exercise more control over every aspect of their daily interaction with the Web. If *My Own Web* is connected to various multimedia and information technologies, it can be used as a the base to Ubiquitous Computing environment in our modern lives.

2.2.3 Service Platform for Rapid Development and Deployment of Context-Aware, Mobile Applications

The service platform presented in [Pok+05] aims to facilitate and speed up the development and deployment of context-aware, integrated mobile speech and data applications. All components are exposed as Web services. The platform provides a dynamic way to develop, deploy and integrate mobile, context-aware services. It also handles many different types of context and offers sophisticated personalization mechanisms to tailor the output of the different services to the current user needs. The platform is domain independent, so it can be used in new or existing applications from multiple domains.

The overall architecture of the solution can be seen in Figure 2.8. Four main groups can be identified in the architecture: third-party services, the platform, the demonstration application and the recommendation service.

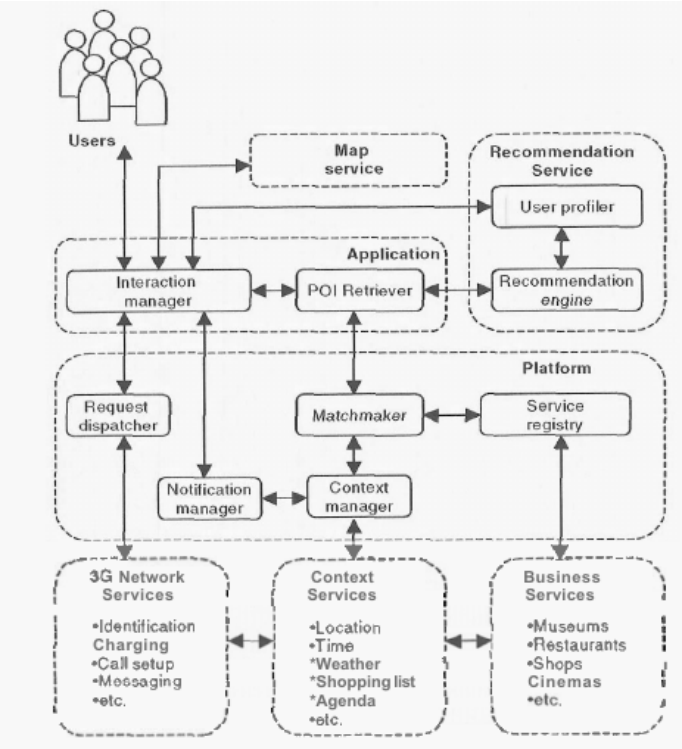


Figure 2.8: Service platform architectural overview [Pok+05]

The multiple core platform components work together to provide the generic and reusable part of the platform. In the third-party services, multiple vital and useful services are present. They are not part of the generic part of the architecture, but are needed

to provide the services. The *Interaction manager* and the *POI*[34] *Retriever* are part of the demo application. Therefore, they are domain specific. The *Recommendation service* has a user profiling module and uses multiple prediction strategies to predict how interesting each POI is for the user. It is not completely generic, but can be reused in similar application domains.

So, this was another attempt at creating a somewhat generic context based platform. Despite some technological limitations, it met some relative success in 2005 and, with technological advancements since then, it should be possible to easily outperform their results.

2.2.4 CXMS - A Context Management Framework

The work presented in [ZSL05] provides a solution for context management, acquisition and processing to be used for application customization. It also covers user profiling and contextualization, providing a full personalization system based on contextual information:

> "an adaptive system (contextualized or personalized or both) follows an adaptation strategy (e.g., pacing or leading) to achieve an adaptation goal (e.g., intuitive information access or easy use of a service). To achieve an adaptation goal, it considers relevant information about the user and the context and adapts relevant system components on the basis of this information."

With those concepts and ideas in mind, they developed a context management framework called CXMS. This framework uses a layered architecture such as the one shown in Figure 2.9. Data becomes semantically enriched step by step while going through the multiple layers, until a decision can be taken about how the domain can be adapted. These layers are:

- **Sensor Layer** - This layer serves as an information collector. Each context-adaptive system relies on a network of sensors placed in the physical environment and delivering an image of the current situation that the user is in.
- **Semantic Layer** - It defines the context model of an application. The context model captures the current situation that the user is in, including her/his preferences, interests, social dependencies, and physical and technical environment. The semantic layer semantically enriches the data delivered by the sensor layer and assigns values to corresponding attributes of the entity's context. The semantic layer completely covers the profiling task of personalization engines and provides different views on the data captured about context.

  In the entity sub-layer, all entities of the domain are defined, and the mapping of sensor data streams to attributes of entities is specified. This layer is also used

[34] Point Of Interest

to specify relations between domain entities and their contexts, and it defines the model for nested entities. Dependencies and relationships between entities are modeled on the entity relationship sub-layer. Those relationships can be dynamically added and deleted. The process sub-layer observes the evolution of the contexts or context parts over time. The application of time series and history modules, statistical models, and intelligent algorithms support this analysis.

- **Control Layer** - Based on the context model and data provided by the semantic layer, the control layer decides what actions should be triggered if particular conditions in the model become true. As a consequence, it generates sequences of commands for the control of the behavior of the domain. The selected action can be seen as an answer to these two questions: *What information is taken into account for adaptation?* and *Which part or functionality of the user interface is adapted, and how?*

- **Indicator/Actuator Layer** - This layer deals with the connection back to the application domain by mapping the decisions taken by the control layer to real world actions. Specialized software components process the delivering of information snippets or the displaying of data on a particular device. As feedback for the control layer, messages indicating the success or failure of actions are sent back.

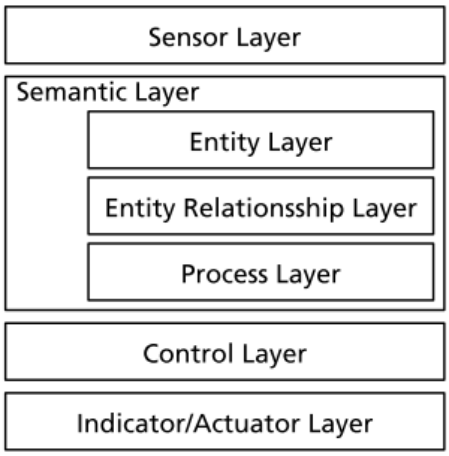


Figure 2.9: CXMS architecture [ZSL05]

Besides the context management system, CXMS also offers additional tools to help developers configure the system to their needs and to integrate their applications with the system.

Another interesting fact about this project is that the authors identified that today's information displays are not always private; therefore, a public display should not present personal information to a group of users standing in front of it. Those displays can be used to filter information relevant to the common characteristics of the physical environment or even take into account user group models. This is somewhat similar to what FCT4U wants to achieve together with P^{2}MUCA. Moreover, the authors even recognize

the potential of combining public and private information displays to yield personalized contextualized contents and services.

Additionally, they demonstrate two case studies that have used their framework to build personalized and contextualized systems. The two examples are an intelligent advertisement board and a museum guide. Both applications adapt their behavior when particular context attributes change their values.

2.2.5 Facebook Open Graph

Facebook has currently available the Open Graph [Fac13] platform. It allows developers to access information about users activity in Facebook and application integration into Facebook. [AGS11] refers some of the things that can be done with Open Graph. There are three main parts that compose Open Graph:

- **Open Graph Protocol**: The Open Graph protocol allows web developers to tag their web pages for the social graph. It enables web site owners to identify their pages as *objects*, such as people, places, activities, groups, organizations, products, and services. Users can then perform actions over those objects;
- **Graph API**: Facebook's Graph API is designed to provide access to every object in Facebook's database: users, photos, videos, statuses, conversations, places, and their relations with each other. Developers use the Graph API to integrate custom applications and use the Facebook as a social data provider;
- **Social plug-ins**: These plug-ins are a simple way to provide integration with Facebook. There are eight individual social plug-ins that can be added to an website by adding a few lines of code.

A good example of the use of these tools is the Levi's website. The strategy was to allow people to *Like* each individual jeans model. Due to the nature of Facebook's social widgets, users visiting the store will immediately see the number of people and which one of their friends *Liked* each jean model, as well as any comments from their social circle. All without logging in or performing any action. Levi's also allowed users to connect to their Facebook profile taking the personalization one step further, sorting and re-arranging the page to prioritize the most popular content within the user's social circle. This can be a good indication of the most valuable content for the user, thus personalizing the site according to his tastes.

Yelp uses Open Graph to deliver personalized search results and recommendations based on friends. It also uses the *Activity Feed* plug-in. This allows the user to see how friends have rated places, what they *Like*, comments they have left, and photos they have added, with recommendations based on that.

Facebook Open Graph helps giving users content that is specifically targeted to their demographic, geography, attitude, behaviors and conversations. Therefore, making their

online experiences more meaningful. On the flip side, consumers are still getting used to this personalized social graph and, while they do it, businesses must find a balance between value and seemingly privacy invasion.

In the case of online stores, Open Graph allows them to evolve to a place where people are shopping in a social setting and using input from others before making purchasing decisions. This mimics real-life behavior. People usually seek opinions, approvals and recommendations before buying. This approach allows to seamlessly integrate that into online shopping.

Facebook provides powerful tools and has the power to change the way we use websites. It is positioning itself as the curator of personal profile data. Preferences can be used across multiple websites and applications. This is also an objective of P^2MUCA. However, there must be caution, implications on privacy are still playing out and it is very important to make sure that consumers have full control over what they share.

2.2.6 Google Personalized Search

Google has some degree of personalization built into its search engine. It is called Google Personalized Search and when a user makes searches while being logged in with a Google Account, all those searches are recorded into Google Web History. Then, when a user performs a search, the results are not only based on the relevancy of each web page to the search terms, but the service also takes into account what websites the user previously visited through search results to choose which results to show next.

Google first released personalized search in late 2005 [Goo13a]. It became available to all users, regardless of being logged in with a Google Account or not, in 2009 [Goo13b]. More recently there has been integration between Google Search and Google+. For example, when a link has been marked as *+1* by a contact in Google+, this is shown in the search results and can also be taken into account when returning search results. Another factor used for personalization is the click through history in a result pages. If a clear preference for a site is shown by being repeatedly clicked when it shows up in results, then its rank may be boosted to appear higher in future results.

It is important to remember that personalization and context are different things. In the case of Google Search, personalization is derived from the user search activity and social connections on Google+. Context is based on other factors, such as language and location.

The presence of this kind of personalization on Google Search has raised some controversy. Even if getting search results related to the user's search patterns and preferences seems a good idea, there may be occasions in which it is undesirable. If a user is trying to find information about a new topic, the personalization of search results may introduce some kind of bias towards what the user usually searches. However, Google tries to avoid over-personalization. Besides that, there may be privacy concerns because Google is logging everything a user searches.

Google Personalized Search may be quite interesting for end users, but it is useless for developing any kind of personalized solution. This could change if some kind of API was made available by Google to tap into these data. However, that could be highly controversial and it must be done with user privacy in mind, if it is ever to be implemented. Nevertheless, it is a good example of personalization being taken into account by a big Internet company.

2.2.7 Other Related Projects

The main identified challenges to personalization are scalability, accuracy, evolving user interests, data collection and preprocessing, integration of multiple data sources, conceptual modeling of the information and privacy concerns [LY05]. Despite all these challenges and some skepticism there have been some progress in this area by major Internet companies (e.g., Google and Facebook) and research projects.

Some of those projects include [GCG05], where a personalization solution to be used on top of Object Oriented Hypermedia (OO-H) is explored. It presents a structured approach to personalize websites based on three criteria: user characteristics (e.g., age or language), user requirements (e.g, intent when visiting a website) and context (e.g., accessing with different devices and from different locations). Users are classified into profile groups. The framework is divided into a user model and a personalization model. All the knowledge about the user is stored in the user model. The personalization model allows the designer to define a collection of rules that can be used to define a personalization strategy for a single user or a group of users.

User profiling techniques based on multiple machine learning techniques are proposed in [KP07]. They have tested their approach using synthetic data sets and some real data (e.g., movie ratings). The results clearly show that with the same data set, different algorithms have a completely different performance. So, it is very important to choose the right algorithms for the right data, taking into account trade-offs between accuracy and time complexity.

Another area of research that can be leveraged for personalization is the Semantic Web. For example, [Abe+10] uses RDF[35], OWL[36] ontologies and SWRL[37] to provide personalization in an e-learning environment.

Just to finalize, a complement to everything that has been said so far is [Zho+12]. It contains a very good overview of all the techniques used for recommender systems, with special detail to the inclusion of social data.

[35]Resource Description Framework
[36]Web Ontology Language
[37]Semantic Web Rule Language

2.3 Interacting with Public Displays

This dissertation also focus on the development of an application to be used in a public space, a public display-based application called **FCT4U**. With the development of this kind of applications comes the opportunity to realize an important long-standing vision of ubiquitous computing, which is the seamless integration of mobile and fixed infrastructures in order to help support everyday tasks [Cli13].

This is a solid opportunity to take advantage of the well-known ubiquity of personal mobile devices and the fact that public situated displays deployment is now less expensive. Those digital displays are becoming increasingly common in different types of companies, residences and public spaces, residing in both indoor and outdoor spaces and being of varying size, shape, form, and purpose.

Public situated displays are invading the urban spaces and we are getting closer to the futuristic vision of Spielberg's Minority Report film [KO13]. Recent advances in display technologies with the consequent falling hardware prices originated the proliferation of (large) public displays, with a great number of them even being connected to the Internet.

Digital displays are an effective way to create ubiquitous presence of digital visual information in the physical world, since they are increasingly pervasive. Large displays are an evident example, with considerable research being done along the years regarding them. Large displays provide several benefits, such as peripheral awareness, productivity and a more immersive experience [Cze+06].

Although this work is not particularly oriented to large displays, it is important to find scenarios of applications with large displays to study interaction techniques to enhance user experience regarding some limitations, such as visual attention [Kha+05] or navigation [FKK07]. Facades displays can be considered as being very large displays, usually presenting low-resolution information to far-away users (spectators), without interactivity and personalization needs. However, exceptions appeared in recent years, such as Climate on the Wall [DH10], which is really an interactive facade in which passers-by can stop to choose words on the facade to be read by far-away spectators.

Thus, taking all of this into consideration, it is important to further explore concepts and projects in this area, that served as an inspiration during the development of FCT4U.

2.3.1 Situated Displays

Situated displays can have an important impact on personal and social behavior, bringing interesting new design considerations and challenges. While there is a growing body of research exploring this area, it remains a complex and normally system specific effort, due to the lack of generalizations for the association between ubiquitous computing applications and display resources.

Such situated displays should enable new and more engaging user experiences by sensing their environment, giving users a more active role in the system behavior, and

providing people with brief encounters with information that is relevant for their specific situation. This could improve local awareness, enrich user perception of reality, promote information sharing within communities, and ultimately make situated displays an integral part of the social settings in which they are integrated.

Despite being a symbol of desktop computing, digital displays can have an entirely new role in ubiquitous computing, providing an important migration path for a paradigm shift. Since they are increasingly pervasive, digital displays of many forms and types are an effective way to create an ubiquitous presence of digital visual information in the physical world. In combination with cameras and other sensors capable of extracting information from the real world, they can then be used to provide contextual information according to the surrounding environment.

Nowadays, displays of multiple types are present everywhere. Some of those displays are:

- LCDs[38] of multiple sizes (e.g., TVs, computer screens, mobile phones and tablets);
- Multiple types of projection displays (e.g., front and rear projectors, mobile/movable projectors and virtual rear projectors);
- E-ink displays, such as the Amazon Kindle Paperwhite[39].

These displays can be seen as situated displays, i.e., displays that are present and embodied in the surrounding environment and being leveraged to display relevant information and provide meaningful interaction, according to the surrounding context. Naturally, the option for the type of display will be according to the environment where it will be placed, application specific needs and cost.

The concept of situated displays provides an execution environment for situated applications, enabling those applications to solicit presentation services from the display, but also to have access to input and context information. There can be two main different approaches to the development of situated and shared computing systems [Pae+04].

On one hand, there is the approach of using big high resolution screens placed in public locations to convey information, like kiosks and digital bulletin boards. This kind of systems can be found in different shapes and sizes, such as street placed screens (e.g., for publicity), projection displays used in conferences and education institutions, and smaller screens placed next to key locations (e.g., a waiting line at a hospital or other public services). The usage of these displays is usually distributed along time and have as input a single physical mechanism (e.g., touch screen).

On the other hand, there is a different approach where the control and access is dispersed among multiple users through the use of mobile phones, tablets and other personal computing devices. This can be used as input devices but also as displays for the

[38] Liquid Crystal Display

[39] `http://www.amazon.com/Kindle-Paperwhite-Resolution-Display-Built-/dp/B007OZNZG0/`

system. The smaller personal devices provide an interaction mechanism for multiple users, but have limited screen sizes which hinders the richness and amount of information that can be displayed. At the same time, situated shared displays provide a larger area where richer information can be shown. Usually, they are also driven by more powerful computers than most handheld devices.

According to their availability and access mode, situated displays may be categorized as follows [Koc05]:

- **Public display**: The display is in a public space, and can be used by all people that have access to it;
- **Shared display**: The display can be viewed/used by more than one user at once;
- **Interactive display**: Users can interact with the display. Concurrent and shared interaction may also be possible/desirable [CJ12];
- **Proactive/Personalized display**: The display can react to the user without any explicit interaction by the latter (e.g., by recognizing users by radio frequency identification and adapting the displayed information to the users).

FCT4U needs an interactive public display. That display must be seen and used by multiple people, while proactively trying to present relevant information under continuously changing conditions. Considering that this application is being developed to validate the personalization platform, the referred proactive aspects will be a perfect testbed to evaluate the ability to personalize the application according to whoever is around the public display installation. It is also desirable that the display can be used by multiple users at the same time. This may be achieved through the integration of personal devices as input mechanisms to the larger shared public display.

2.3.2 Related Projects

In this subsection, a set of research projects in the field of situated public display applications is analyzed and explored. It is only a small selection of the vast amount of references found about the subject, but tries to focus on the projects that seemed more relevant and interesting for the work at hand, while trying to show a broad spectrum that may help to support later design choices.

2.3.2.1 Instant Places

Bluetooth has been used to communicate with nearby displays since the early projects on this topic. This may be due to the fact that currently Bluetooth offers a very low entry barrier to interact with pervasive displays, unlike other approaches that might require the use of not readily available specialized hardware.

An interesting project using Bluetooth is called Instant Places [Jos+08]. It is a research project by University of Minho and Microsoft Research about the use of Bluetooth as a

way to sense the presence of individuals near a public display and enabling a certain degree of interaction. There must be a trade-off between offering richer and active interaction, which usually implies a higher level of appropriation by a single user, and the ability of having a more collaborative social experience with those around.

It was to address this kind of issues that the authors tried to develop a technically simple approach that builds upon the role of presence as the driver for the system's behavior and its situational awareness. By periodically scanning for Bluetooth devices, it is possible to get a continuous flow of presence patterns that may be used as a context source for location-aware interactions. Also, Bluetooth devices can have user defined names which may be used as a simple communication and interaction mechanisms.

In Instant Places, periodic scans are made to collect information about nearby devices. Besides passively detecting the presence of Bluetooth devices, it is encouraged that users use the Bluetooth devices' names as a way to trigger actions in a public display connected to the system. The system does not need any previous knowledge about users, since all the needed information is entirely derived from presence history.

A discoverable Bluetooth device is needed to interact with the application. Once the device is detected by the system it will be shown on the display. However, that person can later switch to a more active role by changing the name of the device. According to the selected device name, users are able to select tagged photos from Flickr, thereby displaying a user generated selection of pictures. The system parses Bluetooth device names in an attempt to recognize commands following a predefined syntax, in order to know which tags the users are interested in.

The system was deployed at a bar in the University of Minho campus for testing and evaluation. The tests showed an increase in the number of visible Bluetooth devices and device name changes, suggesting that there were people that became aware of the system and actively tried to interact with it.

In total, it was estimated that 19.3 percent of the people that had Bluetooth devices used the system. It shows that Bluetooth nowadays offers a very low entry barrier to interact with the system, unlike other approaches that might require the use of not readily available specialized hardware (e.g., RFID[40] tags [McD+08]). However, the use of Bluetooth naming may pose some usability problems, such as the latency induced by passively waiting to be detected or limits imposed to the maximum length of a Bluetooth name.

2.3.2.2 Polar Defense and a Public Display Interaction Framework

Polar Defense is a public display game which was used to study the design of public display applications [Fin+08], [Kav+09]. It is a tower defense game in which people send a set of coordinates though SMS messages to set defensive towers and watch the game taking place in a public display. Using SMS has the advantage that a user does not need

[40] Radio-frequency identification

to be registered to use the system and can just be identified in the screen by a part of the phone number. It also has the disadvantage of a very high latency and it also limits the kind of gameplay that can be achieved.

Besides the game itself, the authors present an interesting view of how people interact with public displays. Based on their research, users in a public display environment are categorized according to their level of interaction and engagement, as seen in Figure 2.10. *Bystanders* are individuals that have no strong interest in the presented content at the display installation. *Spectators* are engaged with the displayed content and surrounding environment, but are not actively manipulating the content on the display. Finally, the *actors* take an active role and may control and manipulate the displayed content.

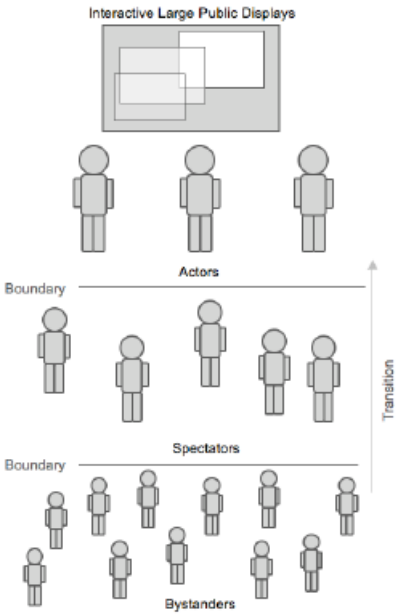


Figure 2.10: Users in a public space

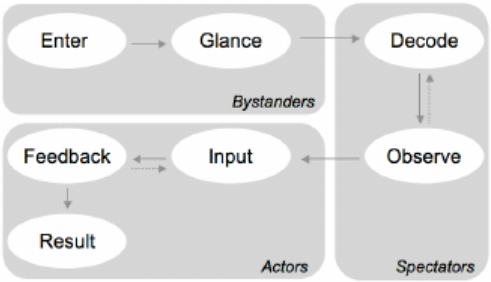


Figure 2.11: Public display interaction framework

Based on this classification, the authors devised an interaction framework to model the user behavior (see Figure 2.11). First, a bystander *enters* a space where a public display is situated. Being aware of the display s/he might *glance* at the display. During this *glance* the user decides whether he has any interest in the content. If so, he may attempt to *decode* the displayed content to understand what is being presented on the display and its purpose. In doing so, the bystander becomes a spectator of the display, because s/he is now somewhat engaged with it. Besides being a spectator, the user may also *observe* other spectators or actors in the environment. If the user decides to interact with the display then becomes an actor. Depending on the interaction model, the user may expect *feedback* to confirm his *input*, which may lead to more user *input*. The cycle continues until the user has obtained the desired *result*.

This view over the different possible users around a public display, their interaction and awareness of the display, is corroborated by other studies (e.g., [BR03], [HKB08], [HM03]).

2.3.2.3 PuReWidgets

PuReWidgets [CJ12] is a programming toolkit for public display. It is an area lacking the sophisticated tools and paradigms that are available for graphical desktop applications. Many public display applications lack consistent interaction mechanisms. This hinders users from using previous experience with public display applications as they do with desktop applications. As such, PuReWidgets aims to provide new tools and abstractions to help building interactive public display applications.

PuReWidgets provides an abstract view of the different display types (from small to really large) and multiple ways to interact with them. It helps developers by abstracting the details of multiple interaction mechanisms that may vary considerably between public display installations. Building these tools entails addressing several requirements. Some of these requirements are common to other interactive systems, but others are very specific to this kind of environment.

Unlike desktop systems, which usually rely on a very small set of input devices (i.e., keyboard and mouse) public display interaction can take advantage of several input mechanisms (e.g., SMS, email, Bluetooth naming, RFID, gesture recognition, face detection, mobile applications, etc.). A single installation cannot be expected to provide all the possible mechanisms, nor it is expected to encounter the same combination everywhere. So, developers should be able to specify their interaction needs in a way that is independent from the interaction modalities that are available at each specific place [CJ12].

It is very important to provide clear and recognizable affordances, enabling potential users to identify the existence of interactive features and how they work across different public displays. Even when facing an application for the first time, the interaction alternatives should be clear, even if the semantics of the operations are not. This follows the basic interface design principle of visibility.

Given the environment in which public display applications usually operate, it cannot be expected that applications are continuously shown and that the ideal screen real state is available. This requires decoupling the affordances for the interactive features from the public display screen itself. As such, the interaction abstractions should transparently allow the interactive features of applications to be rendered in other platforms (e.g., web pages and mobile devices).

Just as with traditional desktop applications, public display applications need a set of controls for developers to choose from, but these controls must be appropriate for public display interaction. As such, PuReWidgets is composed of a widget library and a web service that handles interaction events. A widget abstracts the underlying input mechanism.

The development process is similar to the development of a regular web application. The developer includes an external library in his project and uses the available functions to code the application, instantiating widgets and registering interaction event callback

functions. The library was implemented as a GWT[41] module that developers can include in their own GWT projects. The PuReWidgets service is implemented as a RESTful GAE[42] application. When a widget is instantiated by an application, metadata about the widget is sent to the PuReWidgets service and a remote I/O infrastructure is responsible for accepting raw input events from users. This I/O infrastructure provides abstraction over different input methods. The input events are then routed to the application/widget that was addressed by the user. The service may queue the events until the application is ready to receive them, thus allowing applications to receive events even if they were generated when the application was off. When the PuReWidgets library asks for input, the service replies with the queued input and the library within the application forwards the high-level events to the correct widget instance. This programming model decouples widgets from the application.

PuReWidgets generates and keeps track of widgets by issuing unique identifiers. Some of the available widgets are buttons, list boxes, check boxes, text boxes, number boxes, upload/download controls and check-in controls to announce user presence. These are just abstract concepts, independent of the input mechanisms. PuReWidgets is responsible for supporting multiple input sources transparently. Currently it supports SMS, e-mail, Bluetooth naming, Bluetooth OBEX, QR Codes, mobile and desktop web application.

Besides PuReWidgets, in [Kav+09] there is also a simpler attempt to develop a public display reference architecture to use small screen devices (e.g., mobile phones) as an interaction tool with large public displays.

2.3.2.4 Other Related Projects

There are several other meaningful projects that have investigated how to interact with digital displays, mainly at public spaces. For instance, MAGIC Broker [Erb+08] allows interaction through SMS, the Web, and speech methods. It consists of a middleware toolkit offering separate gateways for each interaction method, allowing parallel usage of several user interfaces.

A similar solution is presented in [Pae+04], consisting of I/O modules for using different interaction techniques simultaneously. It is focused on using mobile devices, which can be used both as interaction devices and as displays for the system itself. The smaller personal devices provide an interaction mechanism for multiple users while situated shared displays provide a larger area where richer information can be shown.

Besides Instant Places [Jos+08], there are other systems using Bluetooth. One of those is the e-Campus system [Dav+09], in which researchers used Bluetooth device names to interact and even to personalize the displays. BluScreen [SPD06] also uses Bluetooth-enabled mobile devices to influence a public display by detecting the audience around it. Users influence the content of a public display only by their detection, thus having

[41]Google Web Toolkit - `https://developers.google.com/web-toolkit/`
[42]Google App Engine - `https://developers.google.com/appengine/`

a passive interaction with the system. The user's passive interaction with the display is co-opted to incidentally provide feedback and thereby change the content a user receives in the future.

Another one is the DiABlu project [Car06] an infrastructure, also based on Bluetooth technology and personal mobile devices, for the detection of people in an environment. It also supports interaction via keystrokes and text messages through a mobile application. The DiABlu server uses Bluetooth to detect nearby devices and to receive messages/keystrokes from these devices, making them available to applications through OSC[43] events. Several case studies in the form of digital art applications are presented in a public display, being dependent on the detection type of the users.

A different and interesting approach to personalize information on public displays is the one applied to Tacita [Kub+13]. That system is comprised of four components in which one of them is an Android application that allows viewers to define content preferences. The first key design decision in this project was to support personalization by having displays broadcast their capabilities to mobile devices, rather than having mobile devices broadcasting user preferences or personal identifiers. This approach is inherently designed to provide a basic level of scalability and user privacy since mobile viewers are not observable by the infrastructure. The second design decision focused on how personalization data is revealed during system use. User requests go directly to the applications that produce the content, and not to the display.

Finally, [KHA12] presents an approach for automatic personalization based on group context. The system recognizes the group of spectators in front of a public display, based on their disposition and gender. It does not require input from the spectator side, neither for training, nor for real-time content adaptation. The experiment conducted in a public area showed that the approach could successfully identify the differences in the content observation of various groups. This is interesting because FCT4U also has some group aware features.

2.3.3 Public Display Interaction Techniques

Multiple options to interact with public displays have been mentioned in the previous related work subsection. This subsection will be focused on studying some of these options to help determine what interaction techniques were to be used by FCT4U.

2.3.3.1 Bluetooth Naming

A possible way to interact with a public display is through the use of some of the capabilities of the Bluetooth technology, namely using a device's address and name. For the problem at hand, we must first understand that a Bluetooth device has a 48-bit unique address and that a user defined device name can be up to 248 bytes long of UTF-8 encoded text. Bluetooth supports scanning of nearby devices through a process called Inquiry.

[43] Open Sound Control

Inquiry employs a frequency hopping scheme. A device wishing to discover others repeatedly transmits inquiry packets on 32 dedicated inquiry channels. At least every 2.56 seconds each Bluetooth device selects one of these frequencies and listens for inquiry requests for 11.25 milliseconds. If an inquiry request is detected it responds through the same channel. The Bluetooth specification states that in an error-free environment an inquiry should last at least 10.24 seconds [Blu07] if all devices in range are to be discovered.

Considering all this, Bluetooth naming can not really sustain real-time communication with multiple devices. By default, when a device is found, name resolution will be attempted. However, this approach is not very good in a busy public location. While it can take less than a second to receive the name from a device, it can take up to 5 seconds to have a name resolution result if the device has moved out of range [Dav+09]. This makes the Bluetooth scanning extremely slow and inaccurate while attempting to keep up with moving people.

An algorithm to alleviate those problems was devised [Dav+09]. It can make things work a bit better by decoupling device finding and name resolution. It also prioritizes name resolution requests, according to the occurrence of the devices' address in the last few scanning cycles. It is still far from real-time interaction, but it allows a more agile detection of Bluetooth device name changes.

2.3.3.2 SMS Messages

Another possible way of a user interact with a public display is through the use of SMS[44] messages [Dav+09], [CJ12], [Fin+08], [Jan+05].

The use of SMS messages may be a good idea, because mobile phones are now extremely common. In fact, according to Pordata [Por13], in Portugal for each 1000 inhabitants there were 1508.9 active mobile phone subscriptions in 2009, while the average of the European Union is 1245.1. This means that mobile phones have a great market penetration.

However, the use of SMS messages have some limitations, such as the maximum length of 160 characters, latency and associated costs for the users.

2.3.3.3 e-mail

E-mail can be used in the same way as SMS messages. However, it supports sending much longer commands and even file attachments. So, it is a good alternative to SMS messages in terms of text-based interaction options. Its use has also been explored in the context of public displays [CJ12], [Jan+05].

According to Pordata [Por13], 68% of all European Union inhabitants accessed Internet at least once a week (51% in the case of Portugal) in 2011. Therefore, when compared with the number of cell phones for each 1000 inhabitants, it seems that e-mail is not as widely available as SMS. How many of those are mobile Internet users is not known.

[44] Short Message Service

This seems to suggest that there are less mobile Internet users than mobile phone users. But besides that, it may be possible to use an open wireless network near the public display that users can use freely to interact through e-mail. Still, just as with SMS messages, the interaction latency may be considerable high due to the unpredictability of e-mail delivery.

2.3.3.4 Mobile Applications

Developing custom mobile applications, whether native or web-based, is an option to interact with public display installations [CJ12], [Kav+09]. A dedicated mobile application offers a platform for richer interaction, because users are able to use their mobile devices to the full extent of its capabilities as a personal input device. This is especially true in the case of native applications.

An application may also enable interaction with better performance and lower latency than e-mail or SMS. However, the real latency gains will vary according to the communication link used between the mobile application and the public display installation (e.g., remote service, directly through Wi-Fi or Bluetooth).

A specific advantage of using native mobile applications may arise if direct access to the device's hardware is desired (e.g., the camera or the accelerometer). However, the installation process may introduce an extra step before the user can actually start to use the application. Users may also be reluctant in trusting the application [Dav+09].

A challenge, when using native applications, is to deal with the difficulty of supporting all relevant mobile platforms. However, for an academic study, supporting a single platform may be enough.

Web applications may be the solution to support multiple platforms and devices at the same time. However, the trade-of is the loss of some performance and platform integration. There are hybrid approaches (e.g., Phonegap[45]) that allow packaging web applications as native applications, offering some abstractions to access native functions.

2.3.3.5 Object and Gesture Tracking

Another possible source of interaction is through the use of some kind of object and/or gesture tracking. For example, a camera that analyzes its surroundings with computer vision algorithms (e.g, tracking hands, faces [CSP09], [Wu+10], [Yua+08] or objects [YJS06]). Specialized motion control solutions can also be considered, such as the Microsoft Kinect [FPT12], [Cox+12], Wii Remote [Sch+08], [FPT12] or Playstation Move.

Using an approach that needs a specific object or controller may be easier to implement and may provide better precision [Cox+12]. However, it may not be practical to assume that users carry a specific object for tracking all the time. As such, a natural user interface may be desirable by using Microsoft Kinect or the ASUS Xtion series[46].

[45] Phonegap - http://phonegap.com/

[46] ASUS Xtion series - `http://www.asus.com/Multimedia/Motion_Sensor_Products/`

Figure 2.12: Motion controllers: a)Kinect; b) Wii Remote; c) PlayStation Move.

It is also important to note that, depending on the used tools, the level of technical difficulty and control precision that can be achieved may vary greatly. For example, using a webcam may be cheaper but it is much harder to implement a satisfactory input mechanism than with a Kinect.

Unfortunately, there are not that many research projects using this kind of approach for public displays. One example is Flashlight Jigsaw, which uses presentation pointers with multiple buttons tracked by a camera system to control a public display game [CMB08]. Another is the Kinect Dressing Room that uses a Kinect controller to make a public virtual dressing room [Kin13].

2.4 Geographical Data Analysis

To build some of the more advanced personalized and context aware features, the system has to capture and analyze user location data. Therefore, this section presents some of the main concepts, techniques and algorithms that can be used to deal with this kind of data.

2.4.1 Geographic Coordinate System

User location data must be collected as points represented on a geographic coordinate system. Those coordinates are usually known as latitude and longitude and are defined over a nearly spherical reference ellipsoid that represents the Earth. The current international standard for such ellipsoid is the WGS84[47].

The latitude (ϕ) of a point on the Earth's surface is the angle between the equatorial plane and a line that passes through that point and is normal to the surface of the reference ellipsoid. The north pole has 90°North (+90°) of latitude, the south pole is 90°South (-90°), and the equator has a latitude of 0°. See the left globe in Figure 2.13.

The longitude (λ) of a point on the Earth's surface is the angle east or west from a reference meridian to another meridian that passes through that point. All meridians are halves of great ellipses, which converge at the north and south poles. The currently widely accepted reference meridian is a line passing near the Royal Observatory, Greenwich, in the United Kingdom. See the right globe on Figure 2.13.

[47]World Geodetic System

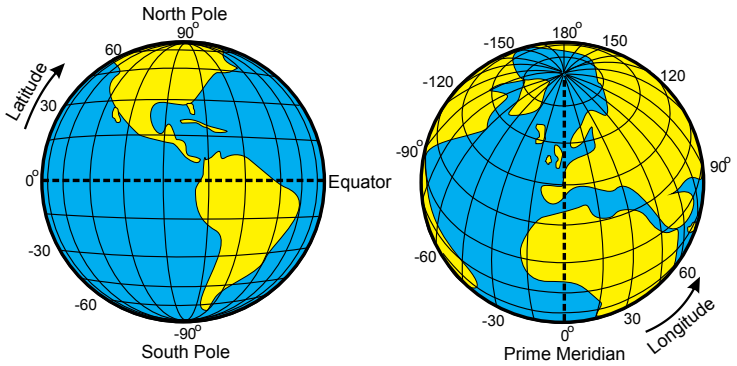


Figure 2.13: Latitude and Longitude of Earth

Both latitude and longitude are usually defined in degrees, minutes and seconds. That representation is usually coupled with S (South) or N (North) for latitude and E (East) or W (West) for longitude. Even though this representation is relatively easy to understand by humans, computers have a difficult time working with it. Therefore, a decimal representation is usually used. This allows each coordinate to be stored as single floating point variable and all mathematical operations can be directly performed on those values. The two different representations for the same point can be seen on Equation 2.1.

$$38^\circ 42' 49.72" N, 9^\circ 8' 21.79" W = 38.713811, -9.139386 \tag{2.1}$$

2.4.2 Distance Between Coordinates

Another important aspect when dealing with geographic information is the need to calculate distances between different points. Such calculations are not completely straightforward. It must be taken into account that the coordinates represent a point on the surface of a spheroid in order to calculate the true distance between two points.

2.4.2.1 Great Circle Distance

One of the simplest ways to calculate the distance on the Earth's surface, with a relatively good accuracy, is to simply consider that the earth is a perfect sphere. Given that Earth is nearly spherical (6378137 meters at the equator and 6356752.3 meters at the poles) this is not a very crude approximation as it can give the distance between points on the Earth's surface correct to roughly 0.34%.

On such case, the distance between two points on the surface of the sphere can be measured as a part of a *great circle*, which is a section of a sphere that contains the diameter of the sphere. Such great circles can be created by planes that intersect a sphere while passing through its center (see Figure 2.14).

These great circles can be used to calculate the distance between any two points that

they pass through. Considering that there is an infinite number of great circles on a given sphere, there is always one that connects two given points on its surface. This notion allows to define a *straight line* in the curved surface of a sphere, often called a *geodesic*.

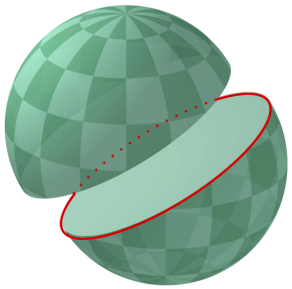


Figure 2.14: Great circle of a sphere

This means that the distance between two points can be defined as the length of the arc of a geodesic connecting the two points. There are multiple formulas that were derived to calculate such length. The simplest one can be seen on (2.2). It derived directly by converting the coordinates from *spherical* to *cartesian coordinates* and applying the definition of the *dot product* to get the angle between the two points and the center of the sphere. The distance between the two points is obtained by multiplying this *central angle* by the radius of the sphere, as seen on (2.4).

Another alternative to calculate the central angle is the *haversine formula* (2.3). This formula is mathematically equivalent to the previous one. However, it works better for small distances because when (2.2) is used on a computer for very close points $\cos \Delta\sigma$ will be close to 1 and thus round-off errors may happen when computing it and the corresponding angle [Dav13].

$$\Delta\sigma = \arccos(\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos\Delta\lambda) \tag{2.2}$$

$$\Delta\sigma = 2\arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cos\phi_2 \sin^2\left(\frac{\Delta\lambda}{2}\right)}\right) \tag{2.3}$$

$$d = r \cdot \Delta\sigma \tag{2.4}$$

Despite being better for small distances, the *harvesine formula* has problems dealing with places that are almost 180°degrees of longitude apart. This again comes from the fact that in such cases $\cos \Delta\sigma$ gets very close to 1 and, once again, rounding errors may occur while deriving an angle from it. However, at such great distances a small error is usually not critical, thus this formula is one of the most commonly used [Dav13].

2.4.2.2 Vincenty's Formulae

Vincenty's formulae are two related iterative methods which can be used to calculate the distance between two points, or the coordinates of another point given an origin point, distance and direction [Vin75]. They are based on the assumption that the figure of the Earth is an oblate spheroid. Therefore, they are potentially more accurate than the methods previously presented, but depending on the desired accuracy they can be relatively slow due to their iterative nature.

The Vincenty's formalae were not further explored because the great circle distance provides more than enough accuracy for this work and due to the algorithm's high time complexity.

2.4.3 Acquiring Location

Geographic data must first be acquired before it can be further processed. Location coordinates can be acquired using GPS[48] or other less accurate methods such as cellular network towers or other wireless networks. Those approaches will be further explored in the remaining of this subsection.

2.4.3.1 Global Positioning System (GPS)

The GPS is a space-based satellite navigation system that provides location and time information in all weather conditions, anywhere on or near the Earth [Nat13]. It is freely accessible to anyone with a GPS receiver.

The GPS project was started in 1973 to overcome the limitations of previous navigation systems [NNC95]. GPS was created and realized by the United States Department of Defense (DoD) becoming fully operational in 1994. The system uses (at least) 24 satellites arranged so that there are at least four satellites in view from virtually any point on Earth (Figure 2.15). This is important because at least four satellites are needed to get a location fix. In general, one can expect an accuracy within 13 meters with a precision of 95%, but real values may vary depending on the receiver, number of available satellites and atmospheric interference [Küp05].

A receiver has to go through the following steps to acquire a location fix:

1. **Identification of satellites** - The receiver must select at least four satellites to be used. The fix will be faster if more information is known beforehand (e.g., the last known position and the satellites' orbits).

2. **Range measurements** - Each satellite emits signals that allow to determine its position, and the time the message was sent. The GPS receiver compares the time the signal was sent with the time the signal was received to determine the distances to the satellites, considering that the signals travel at the speed of light.

[48] Global Positioning System

3. **Position calculation** - The ranges measured are post-processed to correct certain errors (e.g., atmospheric interference). It is then possible to calculate the receiver's position and a time offset through trilateration. This is why four satellites are needed, because there are four equations and four variables. The time offset is used to make corrections due to the less accurate clocks of GPS receivers. The results are then converted to geographic coordinates.

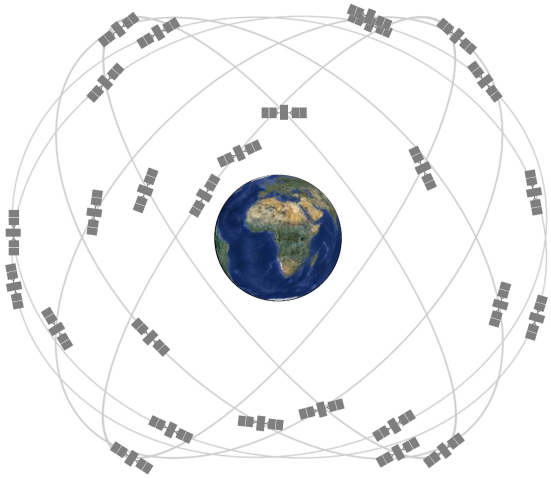


Figure 2.15: GPS constellation [Nat13]

Other Satellite Systems

In addition to GPS, there are other satellite systems in use or under development. The Russian Global Navigation Satellite System (GLONASS) was developed contemporaneously with GPS, but had an incomplete coverage of the globe until recently. There are also Chinese and Indian satellite navigation systems, that currently cover only a part of the globe. The European Union is developing the their own system called Galileo.

2.4.3.2 Cellular-based Positioning

Besides GPS and other satellite-based positioning systems, there has been also some development in system that use cellular networks [Küp05].

The simplest type of cellular-based positioning is through the simple identification of the cell to which a user is connected to by identifying the **Cell-Id**. This offers only a very coarse location. In order to achieve higher accuracy, the Cell-Id may be combined with time measurements to roughly estimate the distance from the cell tower.

More advanced methods were developed to further improve the positioning accuracy. For example, on E-OTD[49] a terminal observes signals emitted by a number of base stations and calculates its position from these measurements through hyperbolic lateration. A method that works in reverse is U-TDoA[50].

[49] Enhanced Observed Time Difference
[50] Uplink Time Difference of Arrival

Cellular networks can also be combined with GPS. In such case, the network provides terminals with assistance and correction data thus helps to significantly increase position accuracy and decrease the TTFF[51]. That method is known as A-GPS[52].

2.4.3.3 Indoor Positioning

Many location based services only work well outdoors. For example, GPS receivers do not work well inside buildings or in close proximity to building that block line of sight to the satellites [Küp05]. However, it is also very important to be able to get location fixes indoors or in close proximity to buildings, as it can happen in an University Campus such as the one in which FCT4U is being tested.

One possible solution for this problem is the use of readily available Wireless Local Area Networks (WLANs) [Küp05], which are pretty common in buildings and are well supported by many devices. This makes WLAN very attractive for serving as a basis for indoor positioning. A WLAN installation inside a building is basically a cellular network that comprises several cells, each served by an access point. Almost all WLAN positioning systems that have been developed so far rely on measurements of the received signal strength (RSS), the received signal-to-noise ratio (SNR), or proximity sensing. Timing measurements are not an option, due to the high synchronization requirements when measuring very small distances. So, in general, WLAN positioning can use one of these methods:

- **Proximity sensing**: The position of the terminal is considered the same as of the access point with the best signal quality. This method can be fairly inaccurate. However, it is the simplest to implement if the location of the access points is known;
- **Lateration**: The position of the terminal is derived from lateration. The distances between access points and terminal are determined by the path loss a beacon experiences during transmission. However, due to multipath propagation caused by the signals being scattered and reflected at walls and ceilings, it is fairly difficult to implement this approach with a good degree of accuracy;
- **Fingerprinting**: The observed RSS patterns originating from or received at several access points are compared with a table of predetermined RSS patterns collected at various positions. The position for which this comparison fits best is then adopted as the terminal's position. As seen in [BP00] and [CK02] this can be further expanded by using signal propagation models in cases. Clustering analysis and probability distributions have been also developed to improve accuracy and reduce computational complexity [YAS03].

One interesting use of these techniques is made by Google. They use fingerprinting and propagation models to provide location-based services for products like Google

[51] Time to First Fix
[52] Assitsted GPS

Maps and Android location APIs. Besides WLAN networks, Google also uses cell tower IDs to assist in determining the location [Goo13i], [Goo13h], [Rap13]. This allows their system to work indoors and outdoors. However, in general, the accuracy of their services vary greatly from zone to zone, depending on the density of WLANs, cell towers and the quality of the data gathered.

There have been attempts at using other techniques for indoor location based services, such as [Küp05]:

- **RFID**[53]: This technology is based on radio signals that are exchanged between an RFID reader and RFID tags. There are active tags with their own power supply, which can be detected at distances of up to tens of meters, while passive tags are powered by the radio signals emmitted by the reader and may only be detected within a few centimeters to a few meters. A tag can be used to determine if a user that is carrying it is within the range of a reader. Or a mobile reader can be used to scan tags that indicate that a user is at a given place. In both cases, the system works as a proximity sensor and it may be useful for some applications (e.g., digital punch clock), but it is of limited use in more general cases.

- **Infrared sensors**: There have been attempts to use infrared sensors do detect the location of infrared emitting badges indoors. The most known example is the Active Badge system [Wan+92] which uses sensors on the ceiling to capture infrared beacons from the badges to determine the location of each user.

- **Ultra sound sensors**: The Active Bat project uses multiple ultrasonic receivers on the ceiling which receive signals sent by a portable emitter [Har+99]. By measuring the time-of-flight, it can calculate the position within the room with an accuracy of a few centimeters. A base station is responsible for coordinating the sensors and the emitter via radio signals. Since sound is much slower, the timings can be used to determine the distance to each receiver which is then used to calculate the location. Cricket is a similar system where the ultra sound emitters are fixed and the listener is mobile and responsible for computing the location [PCB00]. This system also detects orientation [Pri+01].

2.4.4 Identifying Places

Besides getting users' locations, it it is very important to extract meaningful information from that data. For such purpose, clustering algorithms are the ideal choice to group a disperse set of locations into meaningful groups that correspond to a given place (e.g., home, work or the place where a user usually lunches). That kind of knowledge is important to an application that aims to provide contextual information, such as FCT4U.

Considering the multiple types of clustering algorithms, the ones that immediately stand out when dealing with geographical data are the ones based on density. Given a

[53] Radio Frequency Identification

data set with location data collected at a relatively constant interval, it is only natural that places will be identified by a dense cluster of points on a given area. There is no beforehand estimation of how many places a user has visited and there will be many points that do not belong to any place in particular because a user is moving in between places. This is why a more common clustering algorithm, such as *k*-means is not suitable for this case, since it needs to know how many places is it trying to identify and it has no way of disregarding noise.

Also, an advantage of density-based clustering methods is that each cluster can have any arbitrary shape (see Figure 2.16), while *k*-means usually creates clusters of similar spatial extent because the algorithm tries to find spherical clusters that are separable in a way that the mean value of each clusters converges towards its center.

Figure 2.16: Clusters indentified by a density-based clustering algorithm (DBSCAN [Est+96])

So, considering that density-based clustering algorithms seem suitable for the kind of data at hands, some of them will be further explored in this subsection.

2.4.4.1 DBSCAN

One of the most commonly known density-based algorithms is called DBSCAN [Est+96]. The key idea is that for each point of a cluster the neighborhood of a given radius has to contain at least a minimum number of points, thus the density in the neighborhood has to exceed some threshold. Given the generic nature of the algorithm, it can work with multiple distance functions (e.g., Euclidean distance or Great-circle distance) and it can work with data of different dimensionality.

The algorithm needs two input parameters:

- ϵ: The threshold used to define the neighborhood of a point. For a point p its neighboorhood ($N_\epsilon(p)$) are all the other points from the data set for which the distance between p and them is $\leq epsilon$ (including p);
- $MinPts$: The minimum number of points needed to form a cluster.

The DBSCAN algorithm works with the two following concepts:

- **Direct density-reachability**: A point p is directly density-reachable from point q if $p \in N_\epsilon(p) \wedge \mid N_\epsilon(p) \mid \geq MinPts$.

- **Density-reachability**: A point p is density-reachable from point q if there is a sequence of points $p_1 \dots p_n$ such that p_i is directly density-reachable from p_{i+1}.

It is important to note that the relation of density-reachability is not always symmetric. A point p might lie on the edge of the cluster, having not enough neighbors to be considered a core point by itself. However, by starting with another core point q it may be possible to reach p. This leads to the important notion of density-connectivity. Two point p and q are density-connected if there is a point o such that both p and q are density-reachable from o.

So, according to DBSCAN, a cluster is a subset of the original data set that satisfies the following properties:

- All points within the cluster are mutually density-connected;
- If a point is density-connected to any point of the cluster, it is part of the cluster as well.

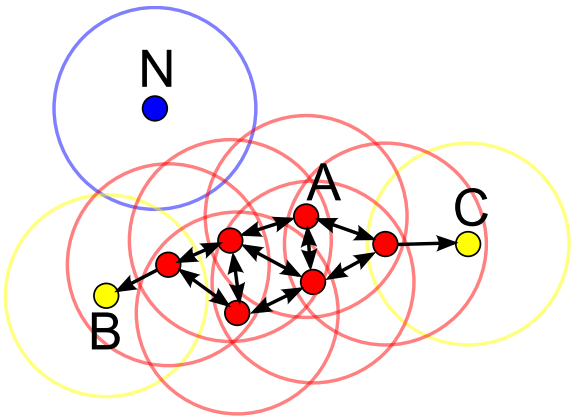


Figure 2.17: DBSCAN clustering concepts: Points at A are core points; Points B and C belong to the same cluster because they are density-reachable from A and thus density-connected; Point N is a noise point when MinPts ≥ 3.

The algorithm starts by selecting an arbitrary point p. $N_\epsilon(p)$ is computed and if there are sufficient points ($N_\epsilon(p) \geq \epsilon$) a cluster is created. Otherwise, the point is considered noise for the time being.

If a point is found to be a core part of a cluster, its $N_\epsilon(p)$ is also part of that cluster. And, if some of their neighbors are also core points, their neighbors are also part of the cluster. The process continues until there is no more points to add to the cluster. At that time, if there are still unprocessed nodes, a new point is selected and the algorithm is repeated from the start until there are no unprocessed nodes. This algorithm can be further understood by the pseudo-code in Listing 2.1.

In terms of time complexity, the algorithm visits each point of the database at least once, i.e., it might visit a point more than once as a candidate for different clusters but only queries for its neighborhood once. Considering that such queries are what mostly

Listing 2.1: DBSCAN Algorithm

```
function dbscan(D, ϵ, MinPts)
   C = 0 //The cluster identifier
   for each unvisited point P in dataset D
      mark P as visited
      if sizeof(N_ϵ(p)) < MinPts
         mark P as NOISE
      else
         C++ //Move to the next cluster
         expandCluster(P, N_ϵ(p), C, ϵ, MinPts)

function expandCluster(P, NeighborPts, C, ϵ, MinPts)
   add P to cluster C
   for each point nP in NeighborPts
      if nP is not visited
         mark nP as visited
         if | N_ϵ(nP) | >= MinPts
            expandCluster(P, N_ϵ(nP), C, ϵ, MinPts)
      if nP is not member of any cluster
         add nP to cluster C
```

impacts the time complexity of the algorithm, if they are done over a linear structure each query will take $O(n)$ time. So the expected overall time complexity is in the order of $O(n^2)$. This can be sped up by using a spatial index. This kind of indexes (e.g., multiple R-tree variants[Est+96] and kd-tree) usually have a search time complexity in the order $O(\log n)$. So if such indexing is used, the overall time complexity drops to $O(n \log n)$.

Finally, in terms of accuracy DBSCAN seems to perform well in many cases. In the original paper it correctly identified all of the 2D spatial clusters [Est+96]. In [Bha09], which makes a comparison between multiple clustering algorithms using spatial data, DBSCAN performed well in terms of recall by retrieving most of the correct clusters. However, it had the highest number of false positives, making it possibly unreliable for some applications. It was also noted that the algorithm has a tendency to join into the same cluster separate places that are very close together, thus having some granularity problems. Nevertheless, it is possible that, by using different parameters, results would have been better in some of the tested data sets, since the parameters were not chosen using an heuristic approach as on other tested algorithms.

2.4.4.2 OPTICS

OPTICS[54] is another algorithm for finding density-based clusters in spatial data [Ank+99]. The basic idea is similar to DBSCAN, but it addresses one of its major setbacks: the problem of choosing a good ϵ for data of varying density.

In order to do so, the points of the database are ordered such that points which are spatially closest become neighbors in the ordering. Through post-processing, this special ordering can be used to extract results equivalent to running DBSCAN with a certain ϵ or

[54] Ordering points to identify the clustering structure

it can be interpreted as a dendrogram that represents the hierarchical structure of clusters with varying ϵ. However, to keep time complexity in check, a maximum value of $epsilon$, called generating-distance, should be provided.

On top of the concepts introduced by DBSCAN, OPTICS defines two new ones:

- **Core-distance**: It is the smallest distance ϵ' between a point p and and an object on its neighborhood, such that p is considered a core point;
- **Reachability-distance**: Considering a point p and a core point o, the reachability-distance of p with regard to o is the smallest distance such that p is directly density-reachable from o.

The algorithm works similarly to DBSCAN, as it starts visiting an unprocessed point and calculating its neighborhood according to the generating-distance and if it is a core point then each of its neighbors will have its reachability-distance calculated. Those neighbors will be added into a priority queue, which stores the next points to be processed ordered by reachability-distance. The points in that queue will then be similarly processed, i.e., its own neighbors can also be added to the queue if they are considered core points.

The end result will be the core and reachability distances of each point in the data set outputted in the order they were processed by the algorithm. Considering that a priority queue is used by the algorithm to process points with lower reachability-distances, points that are close together are usually outputted not far apart. It can be represented graphically as a reachability plot seen at the right side of Figure 2.18.

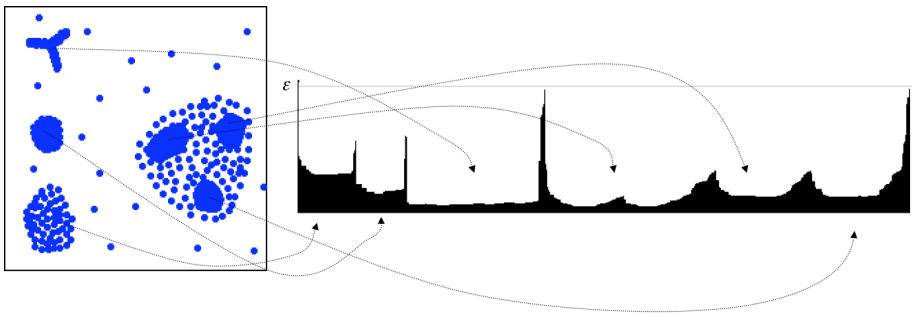


Figure 2.18: OPTICS data visualization [Mue13]

As can be seen in the left side of Figure 2.18, it is possible to analyze the reachability plot to extract clusters. A simple approach is to fix an ϵ value and analyze the graph to find the *valleys* that correspond to the clusters, while also having attention to the values of core-distance to disregard possible noise.

However, a more advanced analysis can be run to build a hierarchical clustering structure where clusters can be nested inside bigger clusters. This is actually the big advantage of OPTICS for application in which a good value for ϵ is impossible to predict due to the variability of the cluster density across the data set.

Finally, in terms of time complexity, just like DBSCAN, the algorithm processes each point once doing a neighbor query for each one. So, as with DBSCAN, a time complexity of $O(n \log n)$ when using spatial indexes is expected, while without them the complexity grows to $O(n^2)$.

However, it is important to notice that even when using spatial indexes, if the generating-distance is set too high it may raise the complexity of the neighbor queries. Additionally, the authors found out that OPTICS has a constant slowdown factor of about 1.6 when compared to DBSCAN, suggesting that the added complexity of calculating core distances, reachability distances and maintaining a priority queue have a relatively big penalty. OPTICS can only be recommended if DBSCAN has some issues with a particular application and OPTICS density structuring capabilities help solve those problems.

2.4.4.3 DJ-Cluster

The final algorithm to be explored is called DJ-Cluster. It was developed specifically for dealing with location data, as presented in [Zho+04] and [Zho+07].

The authors considered the use of DBSCAN, but decided that it had some shortcomings and developed their own approach. DBSCAN was considered to be too sensitive to its parameters values. On top of that, they noticed that for some values of its parameters, the algorithm generates a large number of points within its density definition, and each one of those can be further used to generate its own density-reachable points. That leads to cases where the algorithm used lots of memory and thus slows down considerably. So they decided to create a more efficient algorithm.

DJ-Cluster builds on the concept of connected components and its basic idea is as follows. Just like DBSCAN, it calculates the neighborhood of each point as the points within distance ϵ. If the number of neighbors is less than $MinPts$ than the point is marked as noise. Otherwise, the points create a new cluster, if no neighbor is in an existing cluster, or join an existing cluster, if any of the neighbhors is in an existing cluster. This concept can be easily understood by the illustrations in Figure 2.19. The algorithm itself is pretty simple and can be seen in Listing 2.2.

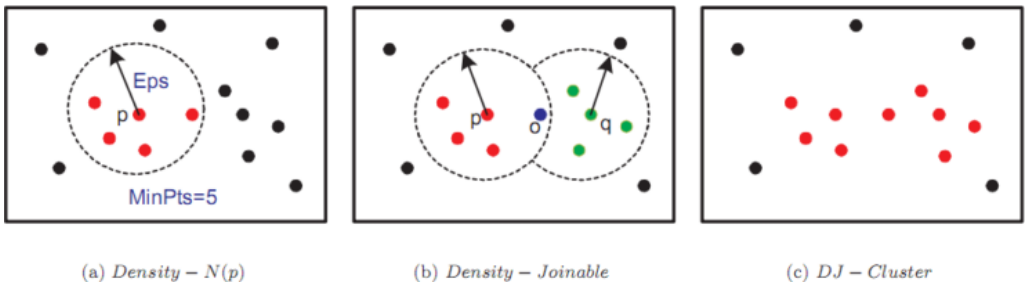


Figure 2.19: DJ-Cluster concepts [Zho+07]

In terms of time complexity, this algorithm has the same base complexity as DBSCAN. It needs to compute the neighbors of each visited point, so it will take $O(n^2)$ time but if

Listing 2.2: DJ-Cluster Algorithm

```
function djcluster(D, ϵ, MinPts)
  C = 0 //Initiate the cluster label C
    for each unvisited point P in dataset D
    if p is null
      return
    if | N_ϵ(p) | == 0
      p is labeled as noise
    else if N_ϵ(p) is density-joinable with at least one existing cluster
      merge N_ϵ(p) and all the density-joinable clusters
    else
      C++
      add N_ϵ(p) to C
```

spatial indexes are used it drops to $O(n \log n)$. Regarding the cluster merging operation, considering that the size of the neighborhood of a point is small and the number of clusters is also small, the cost can be regarded as constant [Zho+07]. Nevertheless, due to the concept of density-joinable being simpler it will tend to use less memory by removing recursive call of DBSCAN's *expandCluster* routine. So the run time is expected to be lower than DBSCAN's.

Regarding sensibility to parameters, according to [Zho+07], the results do not vary that much as long they are set to reasonable values. In their study with geographical data, they have determined that ϵ should be set approximately to the degree of accuracy of the technology being used and that as long as the number $MinPts$ is not very low the number of discovered cluster is about the same.

Both [Zho+07] and [Bha09] conducted tests on DJ-Cluster's recall and precision capabilities. Recall was considerably lower than precision in both cases, suggesting that DJ-Cluster is especially good at avoiding too many false positives. However, it has a tendency to generate false positives on routes between meaningful places [Bha09]. It suffers from granularity problems along with DBSCAN.

So, in general, DJ-Cluster is better suited for applications in which a low number of false positives is required and clusters are relatively further apart. Also, given that it has a lower recall than DBSCAN, it may ignore relevant clusters.

3

P²MUCA - Personalization Platform

Among the main objectives of this dissertation was the development of a personalization platform to help with the creation of personal and context-aware applications.

This chapter will focus on the development of that platform. After introducing previous work that served as a starting point, this chapter will focus on the design decisions and architecture of P²MUCA. At the end, the exposed API (Application Programming Interface) and end user features are presented.

3.1 CAPE - Personalization Framework

P²MUCA uses a component that was previously developed for another Master's dissertation [VC12]. It is called Context-Aware based Personalization Environment and is more commonly known as CAPE. It consists of a machine learning module used together with a generic personalization model which were integrated into P²MUCA. Therefore, it is important to know how CAPE works and better understand the kind of personalization that P²MUCA offers. With that in mind, this section is dedicated to present details about how CAPE personalization model was designed and how machine learning techniques were applied to it.

3.1.1 Requirements

Different users have different interests, different knowledge and different capabilities to interact with digital devices. Personalization becomes much easier to achieve, if it is possible to assign users to different categories that represent the way they behave. CAPE considers this *categorization approach* as user profiling. By collecting data that concerns

how a user interacts with an application, it is possible to use that data in a clustering operation with a defined number of clusters, basically each representing a user category. The result of this operation is a set of clusters/categories where each one contains a collection of points/users. Now that the user profile is known, the application can be personalized to follow each *user's image*.

Supporting a very wide range of personalization possibilities demands a very generic nature, which makes it extremely difficult to use personalization in a completely automated way. CAPE achieves this by having each developer deciding a-priori what parts of the application should be personalized. Data is crucial in this approach and the application's developer has to choose which data should be used in order to create user profiles. For instance, if an application wants to classify its users by level (e.g., basic, intermediate, advanced) it is important to know how much time users spend using the application. On the other hand, data such as time of access may not be relevant for this kind of personalization. This means that the application's developer must study which data resources the application is capable of providing, and choose the resources that will be useful.

Although it seems enough, interaction stream data resources by themselves do not provide interesting results due to their simplicity. The ability to build more complex constructions to represent profiles is desirable. For instance, when using the previously presented notion of user level, it may not be enough to use a single data resource such as the number of logins. An application developer may want to describe the user level as an arithmetic expression composed of variables representing user data, i.e., user level can be defined by a formula like (3.1).

$$w1 * x + w2 * y \tag{3.1}$$

with w, x and y entities being considered as numeric weights and variables. Considering the following values: the first weight to be 0.6; the variable x to be the number of logins; the second weight to be 0.4; the variable y to be the number of clicks on a menu. This would mean that the final value would consist on the number of logins with a weight of 60%, plus the number of clicks on a certain menu with a weight of 40%. This leads to the creation of an abstraction referred in this work as a parameter, which is a bridge between personalization options and resource data, and enables the developer to use data in a more flexible way.

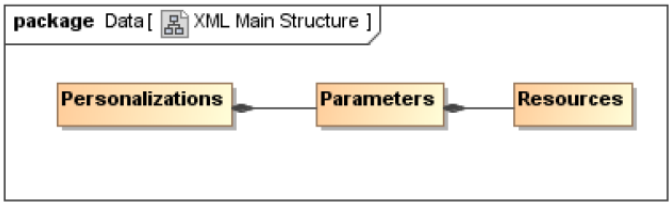


Figure 3.1: Relationships between the main elements/entities

In order to benefit from CAPE, a developer will have to define the desired personalization instances to be used in the application, a set of parameters and resources that are integrated into at least one parameter formula (Figure 3.1).

3.1.2 Data Model

Personalization options, parameters and resources are three main entities of the presented solution, and need to be created and characterized by the application's developer. CAPE's data model can be configured and represented by an XML file, which is processed to initialize the model for a given application. That configuration file is structured in 4 main elements: personalizations, parameters, external services and resources (see Figure 3.2). Given its complexity, the full data model is not presented here, but the following subsections cover all the main components.

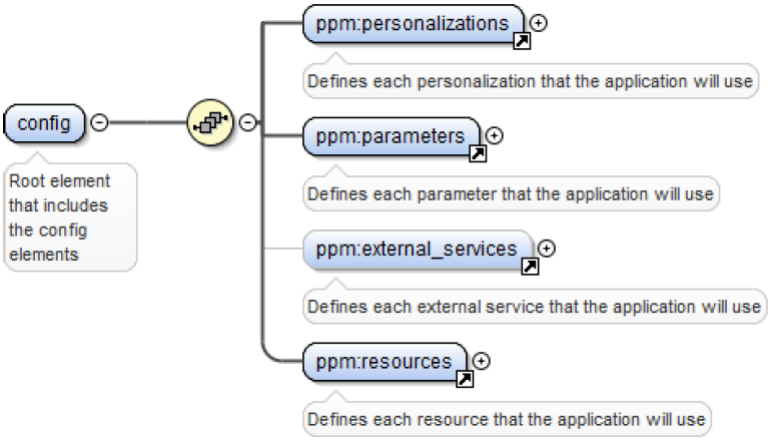


Figure 3.2: Root node of the configuration file

3.1.2.1 Personalizations

The first node defines the personalization instances that will be used in the application or system. Each personalization has a name, a type, the possibility of using context and the listing of all the personalization options to be used. The type element is used for situations where a personalization may use different clustering techniques, but currently only a *k*-means based clustering technique is available.

The next element is optional and concerns to the usage of context in the application (see Figure 3.3). Context has the potential to refine the used data making it more accurate for the profile generation. This type of context is referred along this subsection as context segmentation because it segments data according to the current captured context. When an application submits a request to update data about a certain user (e.g., number of clicks or number of logins), along with the new value, there is also some information about the sending end user's current context (e.g., location or weather conditions). The function that receives this submission updates the respective data values in the database,

but also updates those values under the specific context conditions. For instance, user **A** has 200 logins in total and the application detects that s/he has logged in on a sunny day, so the application will send that update to the personalization service along with information that says it is a sunny day. The service updates the database and changes the data value to 201 logins, but at the same time updates the number of logins the user did, when she was in a context of a sunny day, which obviously needs to be less than or equal to 200. This means that context segmentation can be used to refine the personalization results because in certain situations, the current context has a direct influence on the way a user thinks and interacts.

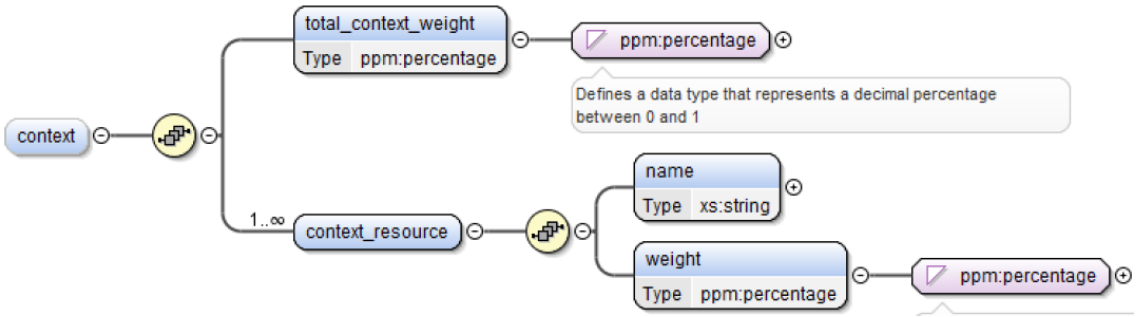


Figure 3.3: Definition of the usage of context in a personalization

Context is composed of another element named total context weight and a set of context resources that have a name and a context weight. The element name specifies what kind of context will be used (current temperature and the current hour of access are examples) and the context weight identifies how much does a context element weight in comparison to other context elements. The sum of the weights of all context resources in a given personalization must always be 1, i.e., each weight indicates a percentage (defines a data type that represents a percentage value between 0 and 1).

When using context, the data itself is modified before generating the user profiles. The operation starts when a personalization request that is sent with information concerning the current context (e.g., time and location). CAPE's database stores resource information that is conditioned by context, and that data is retrieved in order to be weighted with the assigned weights for each context. The result of the weighting operation is multiplied by total context weight value. User data not restricted by context is retrieved from the database and multiplied by $(1 - totalcontextweight)$. The non-contextual result is arithmetically added to the contextual result, forming the final value that will be used for clustering purposes. The following formula (3.2) defines it:

$$[(1 - TotalContextWeight) * (NonContextData)] + \\ + [(TotalContextWeight) * (Wc1 * C1Data + Wc2 * C2Data)] \quad (3.2)$$

Following the optional definition of context comes the personalization options that will be used to define a personalization. Figure 3.4 shows that each personalization option is defined by a list of parameter options and possible external service options.

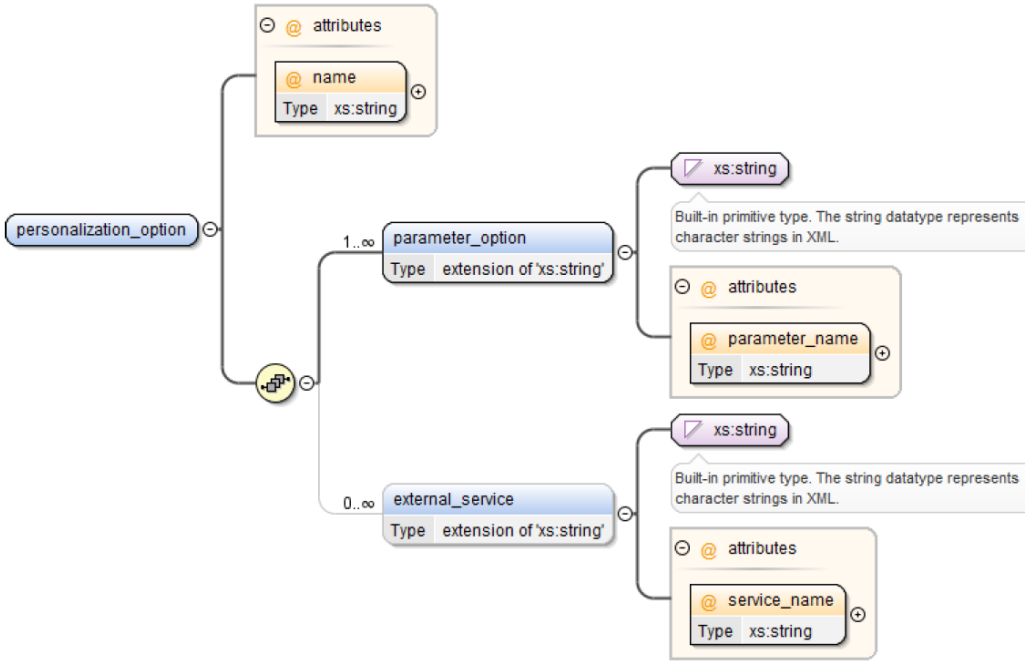


Figure 3.4: Definition of a personalization option

For example, a developer may want to use a personalization in which the options depend on the parameter user level (basic, intermediate or advanced) and on the external service weather (measures the current meteorology condition). These personalization options are the information pool, from which one will be selected and sent to the user as an answer to a personalization request.

3.1.2.2 Parameters

The second node is related to the definition of the parameters and it is presented in Figure 3.5. A parameter is characterized by a name that is used to uniquely identify the parameter and is also used in the definition of personalization options.

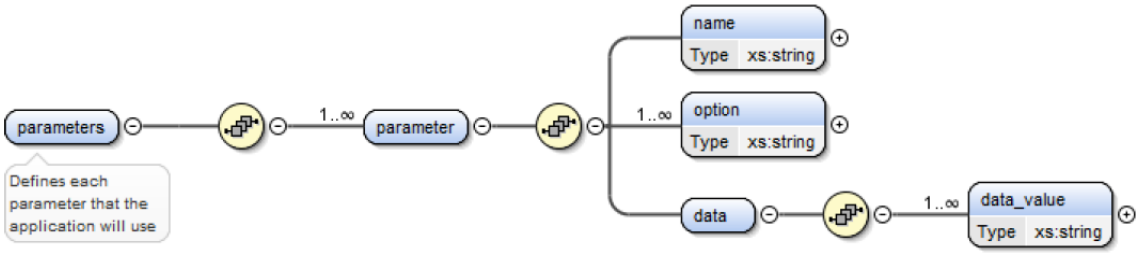


Figure 3.5: All parameters with the formulas and parameter options

This element contains a set of ordered options that identify the clusters that will be used in the clustering process. It is important to point out that the order in which the clusters are defined is crucial to get the correct results. When defining a parameter, the developer must be aware that the first parameter options should be semantically equivalent to lower values, while the last parameter options should be semantically equivalent to higher values. For instance, if a parameter is used to profile users in 3 levels (basic, intermediate and advanced) and the used data consists on the sum of the time each user spent logged-in, then a user with a small value will be considered basic, and another user with a greater value will be considered advanced.

The developer may create one or more different arithmetic formulas in the parameter definition. Those formulas support integer and double type values, basic arithmetic operations (sum, subtraction, multiplication and division) and the usage of parenthesis to express arithmetic priorities between different expressions.

In order to perform the clustering operation, the algorithm needs input, normally in the form of a vector. Each index of that vector represents a user. Each parameter formula represents a different dimension in the vector of data that will be used in the clustering process, i.e., instead of having a vector of users with only one value used in the clustering process, a matrix of users is used, where each user may have more than one value, which concern to other different formulas.

3.1.2.3 External Services

This third node is used to define which external services will be used in the application. As with the parameters, a name element is used to identify the external service when defining the personalization options. The second element is used to classify the type of the external service, selected from a collection of limited types provided by CAPE. It can be extended to use multiple external services, but so far the only external service is related to meteorology using the web-service OpenWeatherMap[1]. The last element aims to forearm situations when the external service is unavailable. A default value is defined depending on the type of external service, i.e., the element is coincident with one of the available external service options. For instance, if the weather service was unavailable and a personalization depended on it, the later would use the predefined default value (e.g., *sun*).

3.1.2.4 Resources

The last node marks application's resources, which corresponds to another main entity in our model. CAPE considers three types of resources: interaction stream, context and preferences (Figure 3.6).

The first element referred as resource_appdata addresses interaction stream data, i.e., data obtained from the normal interaction of each user with the application. There can be

[1]OpenWeatherMap - `http://openweathermap.org/`

different types of data updates. Therefore, the developer must specify, from a limited list, what kind of update type will be used for a given resource. CAPE initially supported two update types: increments of one unit and addition of values. However, a replace update type, which allows the resource value to be completely overwritten, was added during the development of P²MUCA.

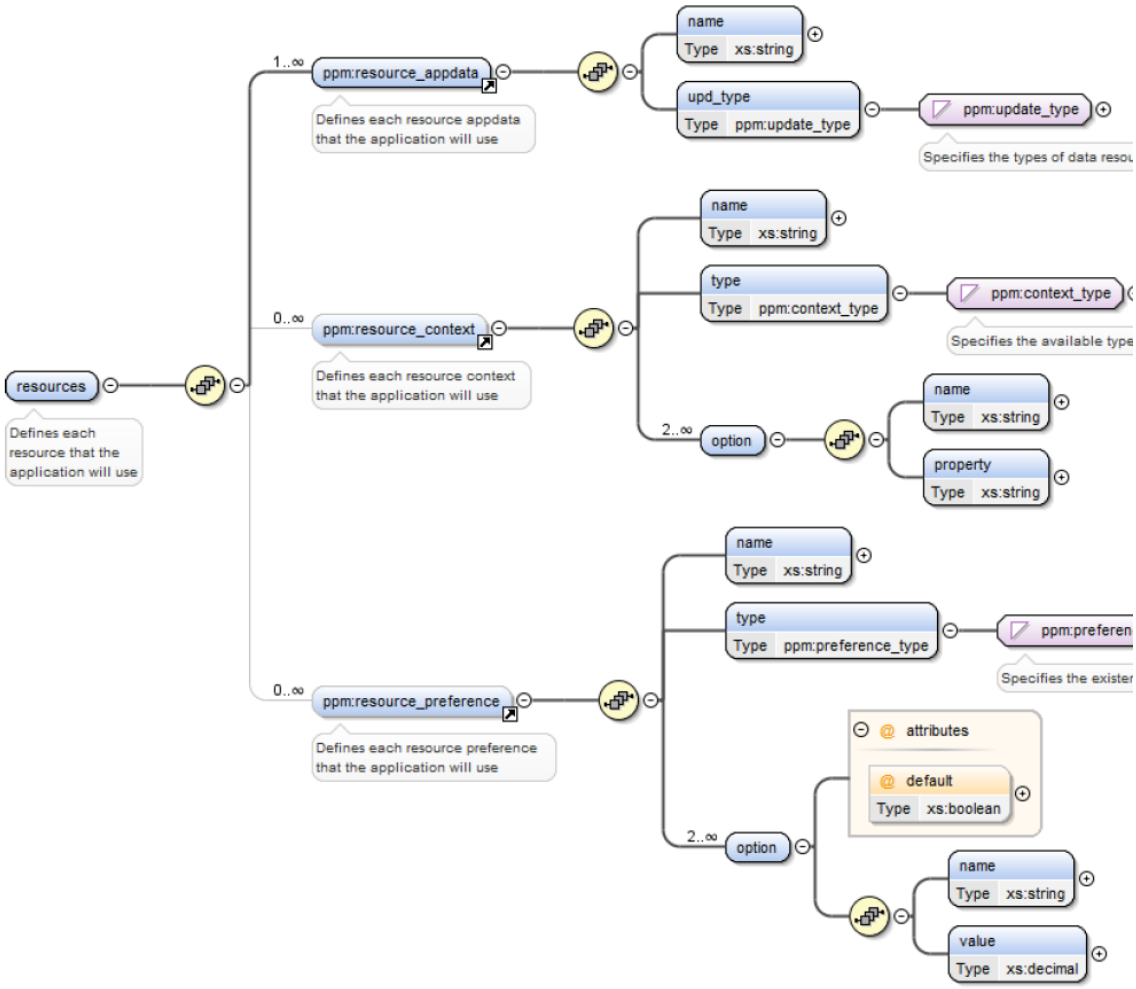


Figure 3.6: All parameters with the formulas and parameter options

The second element addresses data concerning the context of the application's usage. Each context resource has a name that identifies it and also a type to describe it. Following that, there is a set of options that depend on the chosen resource context type. Those options are used to describe how many context options will be used in any personalization process that uses context segmentation.

The types of context resources initially supported by CAPE were:

- **hour_interval** - Each context option should have a property that expresses the hour intervals, like 08:00-14:00. This context type is used based on the current user's time.

- **temperature** - Each context option should have a property that expresses the temperature intervals, like 21-40. This context type is powered by OpenWeatherMap based on a user's location.

Additional context types were added during the development of P²MUCA to extend the use of context resources to any possible scenario. Developers can use these generic types of context resources in cases that are not covered by the more specific types. These new types are:

- **interval** - Each context option should have a property that expresses any arbitrary numeric interval, e.g., 0-5. This value is passed directly as part of a user's context as a key with the name of the resource, e.g., the key *score* with the value 3.
- **value** - Each context option should have a property that expresses any arbitrary value, e.g., numeric or textual. As with the previous context type, the value is passed directly as part of a user's context as a key with the name of the resource.

The last resource type is dedicated to user preferences. It has a name element to identify the preference, a type of preference like open-text or multiple-choice, and a set of options that may be used to describe the accepted values. Each option has a name that identifies it, a numeric value and a boolean attribute that indicates if the option is to be used by default.

The value element can be used arithmetically (sum, subtraction, multiplication and division) in parameter formulas just like any other resource. For example, assuming that variable $pref1$ is a multiple-choice question and the user chooses an option with value 1.1, the formula (3.3) would suffer a numeric increase of 10% which helps refining the parameter value.

$$parameterDataFormula = timeLoggedIn * pref1 \tag{3.3}$$

3.1.3 Personalization Algorithm

CAPE's personalization algorithm uses data from a database that follows the model previously described. This database is configured through the loading of an XML file with the structure shown in Figures 3.2 to 3.6. A summary of the algorithm using the data model to provide personalization to an application is presented as follows.

Upon receiving the request from an application on behalf of a user, having her/his identification, the personalization identification and context information attached to the user's request, the algorithm simply consults the parameters that are being used for the requested personalization, and for every user in the application it parses the formulas in each parameter. This results in a matrix of users-values where each line represents a user and each column represents a data dimension. This means that if there are, for example, three formulas in a parameter, the clustering process will have three dimensions. The

matrix is used as input for the clustering process, which outputs the respective clusters for each user.

The clusters have a direct correlation with the parameter options, meaning that the combination of clusters resulting from clustering every parameter used in the application will result in a single personalization option. User profiles are generated by a machine learning module using the *k*-means clustering algorithm.

For a personalization including context segmentation, the data structure of users and respective parameter formula values is still populated in the same way as previously, but instead of sending it immediately to the clustering operation, the algorithm makes some modifications to the user data. For each context option that the requesting application was assigned (being alone, or at work are examples of different context options) instead of getting the normal user's interaction stream from the database, retrieved data is conditioned by the user's current context.

In case the personalization addresses any external service (post-algorithm filtering), the external service will be consulted and the personalization option along with the external service's response will be retrieved to the user as a response for the personalization request.

CAPE already had a basic personalization API to test the aforementioned model and personalization algorithm. However, it was mostly developed as a proof of concept to validate the personalization model. Therefore, P²MUCA uses CAPE as its base model but replaces the orginal testing API to provide a more feature rich service that can be publicly available, thus focusing on availability, ease of use and security.

3.2 Design Decisions

This section presents some of the more important design choices that were made to build P²MUCA. Those decisions cover areas such as the programming language, development platform, the chosen type of web services, data storage and the authorization mechanism used to secure the service.

3.2.1 Programming Language and Platform

Considering that CAPE was mainly written in Java, it made perfect sense to continue using Java-based tools to ease the integration process. Java EE[2] technologies are the standard tools to implement web applications and services in Java. They are just a set of standards for multiple purposes, not being tied to any particular implementation. However, GlassFish[3] is considered to be the reference implementation.

It would be possible to implement P²MUCA using only Java EE standards. However, there are other non-standard tools that integrate very well with the Java EE stack. One of

[2]Java Platform, Enterprise Edition
[3]GlassFish - `http://glassfish.java.net/`

them is Spring, a Java-based development framework by SpringSource[4]. One of Spring's subproducts is Spring Roo[5], which is based on the extensive set of Spring family tools, such as Spring MVC and Spring Security[6].

Spring Roo is an interactive shell that enables agile development by helping to configure the project through the use of declarative commands and automatic generation of boilerplate code. This code generation is made through the use of an aspect oriented [Kic+97] extension of the Java programming language called AspectJ [7]. Also, Spring Roo can be easily integrated with Eclipse[8] through the use of the Spring Tool Suite[9], thus providing a full development environment.

3.2.2 Service Type, Authentication and Authorization

Regarding the kind of web service technology to use, it was soon decided that P²MUCA would be implemented as a simple HTTP-based service. This is justified by being a simpler service model that is easier to integrate into third-party applications, that only need to be able to send HTTP requests. Also, it is possible to extend the current design to support WS-* services if that option proves to be advantageous to third-party developers using the platform.

Other important aspects for the developed service are the authentication and authorization mechanisms. While developers should be able to get access to user data and the functions offered by the platform, users still must have control over what applications have access to their data. This is very important, because a personalization framework may hold privacy sensitive data that must be protected from unauthorized access.

As mentioned in 2.1.2, there are multiple standards that can be used to deal with authentication and authorization. Given that the major problem here is authorization, and authentication may be done with a simple login form on the P²MUCA, OAuth 2.0 was selected as the authorization mechanism. More complex standards could have been used, but considering the ease of use aspect and the fact that the P²MUCA's API is to be used in the open web and not on an enterprise environment, it made sense choosing OAuth which is also used by major Internet web sites and applications.

An extension to Spring Security called Spring Security OAuth[10] was used to implement the server-side of the OAuth protocol. This library supports both OAuth 1.0 and 2.0, but only version 2.0 was used because it is simpler to work with and it should provide more than enough security when used over HTTPS.

[4]SpringSource - `http://www.springsource.org/`
[5]Spring Roo - `http://www.springsource.org/spring-roo`
[6]Spring Security - `http://static.springsource.org/spring-security/site/`
[7]AspectJ - http://www.eclipse.org/aspectj/
[8]Eclipse IDE (Integrated Development Environment) - `http://www.eclipse.org/`
[9]Spring Tool Suite - `http://www.springsource.org/sts`
[10]Spring Security OAuth - http://www.springsource.org/spring-security-oauth

3.2.3 Data Storage

Regarding data storage, the obvious choice was to use a relational database. MySQL was chosen for development, which is easily available and has proved to be powerful enough for relatively complex deployments [Ora13b], [Twi13]. Nonetheless, considering that the service is being built using the database abstraction provided by a JPA[11] implementation, it is trivial to change the DBMS[12] that is being used. However, since CAPE was developed to use a MySQL database, it might not be possible to easily switch to another database system without rewriting some parts that use non-standard SQL[13].

3.3 Architecture

P²MUCA's has been developed as two major components:

- **P²MUCA website** - A front-end for creating new applications and configuring the underlying personalization model provided by CAPE. It is also used as a user sign-in and authorization point, allowing users grant and revoke an application's access to their data.
- **P²MUCA service** - Another component that implements the HTTP service API itself protected using OAuth 2.0. It uses CAPE to support all of the provided API calls.

Both components depend on each other. For example, P²MUCA website allows the configuration of the application personalization model through CAPE, but that same definition must be shared with the P²MUCA service. Also, the OAuth data must be shared between both, because P²MUCA website is responsible for creating applications' client credentials and access tokens, but the P²MUCA service endpoint must use that same data to authorize each request made by third-party clients. Therefore, two databases are used:

- A database to store information about P²MUCA users, applications and OAuth data;
- The CAPE database that needs to be manipulated by the two components.

A schematic view of the architecture is shown in Figure 3.7. As can be seen, the P²MUCA service simply offers an HTTP based API secured behind an OAuth 2.0 Resource Service. This Resource Service must have access to the P²MUCA database because there is data that must be shared with the OAuth 2.0 Authorization Service running on the P²MUCA website. It is implemented as a Spring Security filter with the aid of the Spring Security OAuth library.

[11] Java Persistence API
[12] Database Management System
[13] Structured Query Language - A special-purpose declarative programming language designed for managing data held in relational databases

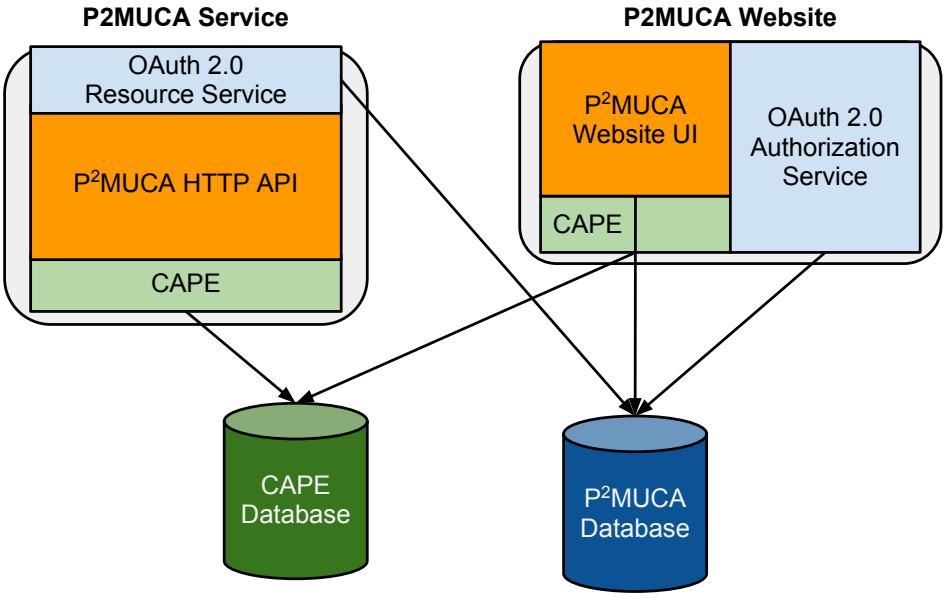


Figure 3.7: P²MUCA architectural overview

As already said, the use of the OAuth protocol protects data from unauthorized access. This comes from the fact that an application must first get an access token and a refresh token to use the personalization API, and those can only be obtained if a user authorizes it. The access token can then be used by a pre-determined amount of time to make API requests on behalf of an user, while the refresh token can be used to get a new access token if the original one expires. A user can always revoke an application access and both the access and refresh token are rendered useless, thus giving a user express control over who can access their data.

Two common scenarios regarding the use of OAuth 2.0 can be seen in Figures 3.8 and 3.9. The first is the *authorization code grant flow*, where an application asks for permissions from the user using its *client id* and gets an authorization code that can be exchanged for access and refresh tokens by providing its *client id* and *client secret*. As the access token can expire after a given time, the refresh token can be used to get a new authorization token without the user having to go through the authorization page once again.

The second flow is called *implicit grant flow* and should be used in cases where the *client secret* can not be used because it is impossible to keep it private (e.g., a browser based application or native applications), while also providing an alternative simpler flow. This way, only a short lived access token is issued as part of the redirect URI with no companion refresh token. Once the access token expires, a new one must be requested which means that the user must once again be redirected to the authorization URI. However, if the application is already authorized and the user is still logged in, the user should be automatically redirected back to the application and the new access token delivered to it through the redirect URI.

It is also possible to exchange the client credentials (*client id* and *client secret*) for an access token that can be used to make requests on behalf of the client itself. This flow is depicted in Figure 3.10.

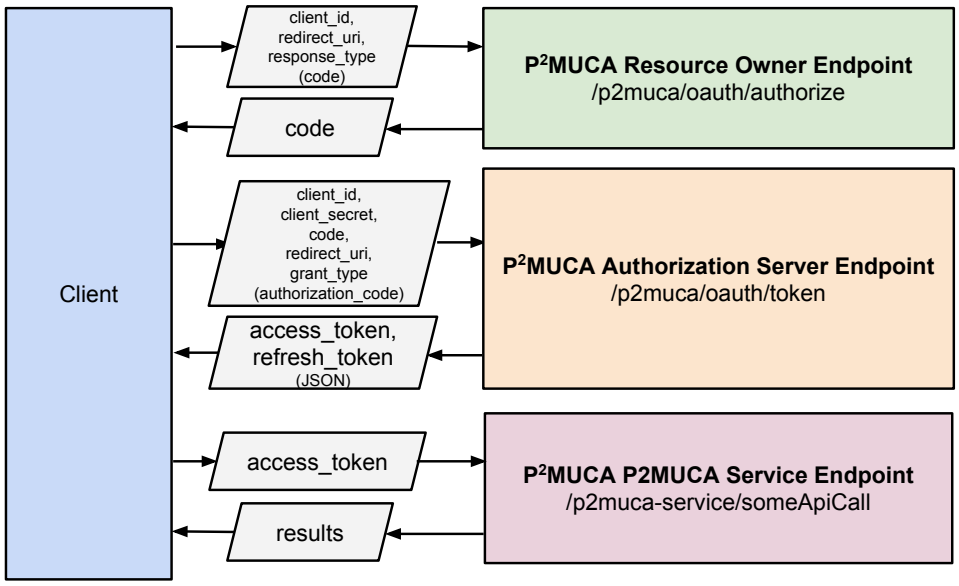


Figure 3.8: P²MUCA using OAuth 2.0 Authorization Code grant flow

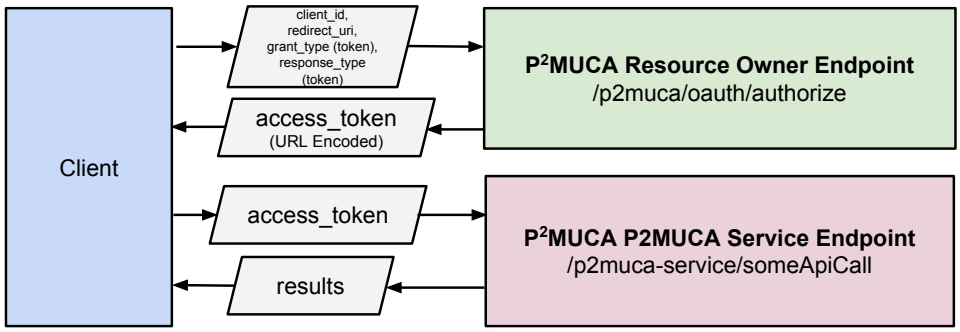


Figure 3.9: P²MUCA using OAuth 2.0 Implicit Grant flow

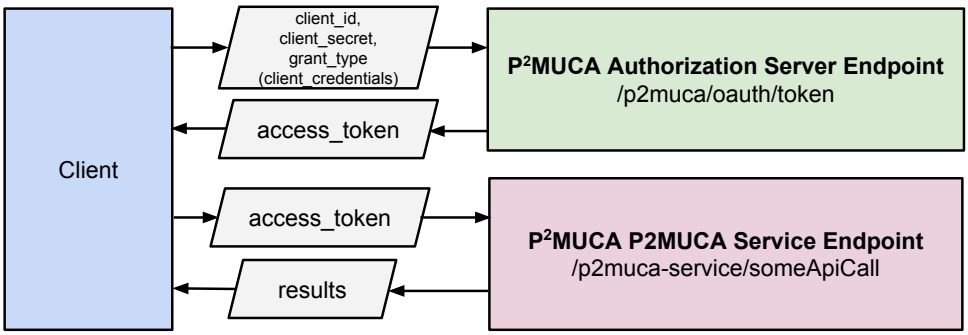


Figure 3.10: P²MUCA using OAuth 2.0 Client Credentials Grant flow

The exposed API is a direct mapping of all the operations offered by CAPE. This is why it stands on top of CAPE on the architectural overview. The HTTP API simply wraps those operations and exposes them through a simple HTTP endpoint secured with OAuth, which can be used by any application developer just by issuing simple HTTP requests. Then, CAPE than needs to communicate with the database that has all of the tables and data that support its meta-model [VC12].

P²MUCA website also needs access to the two databases because it also uses CAPE. When a developer is creating or editing settings it must use the CAPE configuration module to apply changes which will be recorded in CAPE's database. This is done through the Website UI either by submitting a CAPE XML configuration file or by using a newly developed P2MUCA configurator (presented later on 3.6.1). Besides that, the Website UI also needs to access the P²MUCA to manipulate the OAuth data while creating or editing applications and when users give or revoke application permission.

Additionally, the P²MUCA database also stores user authentication information, such as usernames and passwords of the end users and third-party developers, so that they can authenticate with the website and manage their applications and permissions, respectively.

The OAuth 2.0 Authentication Service is part of the P²MUCA website. Just like the Resource Service, it was implemented thanks to the Spring Security OAuth library. It is responsible for the authorization endpoint and the token grant endpoint. The first is used when users are asked whether they authorize a certain application, and the second returns access and refresh tokens when one of the multiple grant types defined is used [Int12b].

Moreover, a validation endpoint was implemented so that developers can use OAuth tokens for authentication. It validates the owner of that token and which client application requested the token, so that an application can be assured that is valid and belongs to the user who the application is dealing with. This is a very simple mechanism, but in fact OpenID has already been extended to be used with OAuth 2.0 in a similar way in a draft standard called OpenID Connect [Ope13]. It may be implemented in future work since it is a more standard approach that offers extra security.

3.4 Deployment

The two components were developed as Java Servlets so that they could be easily deployed on any appropriate cloud platform. During development, both components were initially hosted on the Cloud Foundry public PaaS service, besides being tested locally on a Tomcat[14] installation. However, that service became paid with only a time-limited trial available.

[14] Apache Tomcat is a Java-based web server and servlet contained - `http://tomcat.apache.org/`

Considering that Cloud Foundry is an open PaaS there are multiple providers that offer it. So another Cloud Foundry provider, which offers a free plan, called AppFog[15] was found. Since they are based on the PaaS software, the migration from Cloud Foundry's public service to AppFog's was trivial. Changing the deployment target from URL to another was almost all that was done. However, AppFog started to have some problems. It suddenly started to undeploy the applications randomly and it was not possible to determine the cause of the problem whilst no direct support is given by the provider on free accounts.

So, as a last resort, P²MUCA was migrated to OpenShift by Red Hat that offers a free plan. The migration was not that easy this time, because the set of tools used for Cloud Foundry-based provider could no longer be used. In fact, OpenShift offers a much more bare bones approach where developers can readily change the default configuration of a pre-defined application template, or even start one themselves. It even offers SSH[16] access to application instances. However, such flexibility comes at the cost of added configuration complexity.

Each deployed application on OpenShift is hosted through a Git repository with a predetermined directory structure. This repository may contain a ready to run WAR[17] file or a Maven[18] project ready to be compiled and run. The second option was chosen since the components are already being developed as Maven projects.

The database configuration is more complicated. While on Cloud Foundry it is possible to make an automatic database configuration just by inspecting the data source configuration, on OpenShift a full alternate configuration must be provided to be used on deployment. That configuration must be chosen by the platform when launching the Maven project, thus a special build profile is needed to choose it when the application is run on OpenShift.

Still related to the database configuration is the fact that on OpenShift each application has their own databases associated to it. However, P²MUCA components, which are deployed as two distinct applications, must have access to the same databases. This use case is not very well supported under OpenShift, but there are two options:

- **Scaled Applications** - Scaled applications have their databases running as separate instances, so that they can be reached from the multiple concurrent application instances that are used to balance the load. However, it is also possible to connect to those databases from other applications.
- **SSH Tunneling** - Another possibility is to add a database to a regular application instance and, from another application that needs to access the same database, make an SSH tunnel through OpenShift's internal network. This bypasses the firewall that would normally not let make a direct connection to a database running on

[15] AppFog Platform as a Service - https://www.appfog.com/
[16] Secure Shell
[17] A WAR file (**W**eb application **AR**chive) is an archive file used to distribute a Web application.
[18] Apache Maven - `http://maven.apache.org/`

another application instance.

At least 4 gears (application instances) would be needed for the first option but the free plan only offers 3 gears. The second option introduces network overhead due to the use of tunnels and the need of extra configuration to configure them. Despite those flaws, the second solution was adopted because only 2 gears are needed, making it possible to implement it on the free plan.

Figure 3.11 should help to understand how the deployment was made on OpenShift. Each of the main components was deployed on its own gear. A database was created a database was created on each of those gears, but direct communication to the database can only be done within the same gear. So, P²MUCA Website needs to establish an SSH tunnel to the other gear to access the CAPE Database. Similarly, P²MUCA needs a tunnel to the P²MUCA Database.

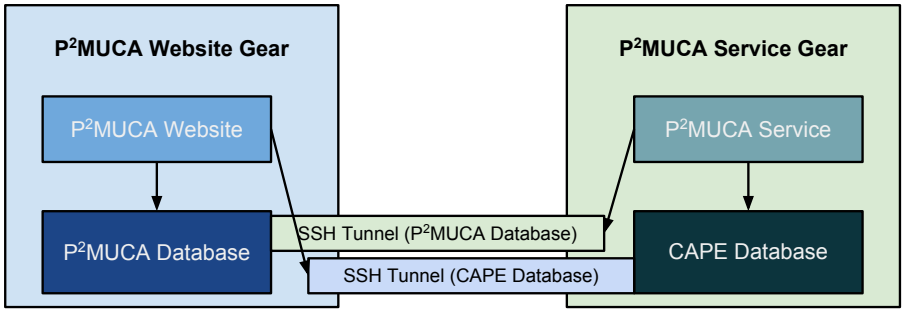


Figure 3.11: P²MUCA deployment on OpenShift

In a real production environment it is recommended to switch to a paid plan and use scaled applications. That approach is more elegant and provides the possibility of scaling the P²MUCA Website and Service (independently) if needed.

3.5 P²MUCA HTTP API

As already stated, the P²MUCA service was developed as an HTTP-based web API. It follows more of a remote procedure call pattern than a more standard RESTful service because the provided operations are not closely mappable by such an approach.

The provided API calls were based on the ones already made available through some PHP scripts that connected to CAPE's Java code [VC12]. However, those were further developed while a few new ones were added when it made sense. In all cases, they were protected using OAuth 2.0.

All of the supported API calls are listed in Table 3.1. Note that an access token must be passed in all cases as an URL parameter with the key *access_token* or as an HTTP authorization header with the keyword **Bearer** followed by the token. The access token must be acquired by using one of the already presented grant flows. Also, whenever *context* is used as a parameter it means that a JSON object with string keys and string

values with information about a user's context should be passed. Just like in the initial CAPE version, such context can contain information about the current time for a user or the geographical coordinates of where s/he is.

Two additional context types were added, the interval and value. Context resources can be defined as having one of those types and can be passed in the context map, thus allowing the third-party developers to define and manage new context types that are not built-in into the platform. The interval type allows a numeric interval with a '-' as separator (e.g., 5-10), while the value type just matches any passed string. An example of the value type would be the context with the key *place* and the value *home*, indicating that the user is currently at his home. A simple example of a context map that covers all of the supported types can be seen on Listing 3.1.

Listing 3.1: JSON-econded context

```
{
  time: "15:37",
  latitude:  38.708056, longitude: -9.138333,
  intervalName: "5-10",
  valueName: "atHome"
}
```

Note that all but one of the API calls are user specific. In such cases, the user is identified based on the access token. The only exception is the *addUser* API call for which a client specific token should be used, i.e., a token should be obtained for the client itself through the client credentials authorization grant flow [Int12b].

Name	**Method**	**Input**	**Output**
/validate	GET	*No parameters, just the access token*	JSON object with the corresponding *clientId* and *username* of the given access token.
/addUser	POST	username; email. (*Note*: The access token must be an application specific token instead of one obtained from a user)	JSON object with a *status* property equal to *success* or *error* and a *message* property with the status message.

Name	Method	Input	Output
/getPreference	GET	preference	JSON object with a *status* property equal to *success* or *error* and a *message* property with the status message. If successful the resource value is returned under the *result* property.
/setPreference	POST	preference; value	JSON object with a *status* property equal to *success* or *error* and a *message* property with the status message.
/setPreferenceValue	POST	preference; value	JSON object with a *status* property equal to *success* or *error* and a *message* property with the status message.
/getResource	GET	resource; context (*optional*)	JSON object with a *status* property equal to *success* or *error* and a *message* property with the status message. If successful the resource value is returned under the *result* property.
/setResource	POST	resource; value; context (*optional*)	JSON object with a *status* property equal to *success* or *error* and a *message* property with the status message.
/getPersonalization	POST	personalization; context (*optional*)	JSON object with a *status* property equal to *success* or *error* and a *message* property with the status message. If successful the personalization option the user falls in is returned under the *result* property.

Table 3.1: P²MUCA HTTP API calls

3.6 User Interface

P²MUCA's user interface consists of a website for users and developers. Basically, users can register, create applications and manage permissions given to third-party applications.

The user registration and login can be seen in Figures 3.12 and 3.13, respectively.

Note that the system supports login through Facebook and Twitter credentials, if the user associates a Facebook and/or Twitter account with their P²MUCA user. The registration process can also be triggered by using one of those external accounts, which means that some of the data will be automatically filled in the registration form and the new user will automatically have the external account associated to his/her profile. Just like in many other websites, a confirmation e-mail is sent to the newly registered user with an activation link. That link must be followed to activate the account before the user can login into the website using the form on Figure 3.13.

Figure 3.12: P²MUCA user registration

The two main features available to users are: 1) managing applications that use the personalization model provided by the platform; and 2) managing permissions given to third-party applications. Those options are available to any user. There is currently no distinction between end users and third-party developers. This is intended, because a developer may also be a user of other applications, or of their own. The only downside is that regular users are presented with additional options that they may never use.

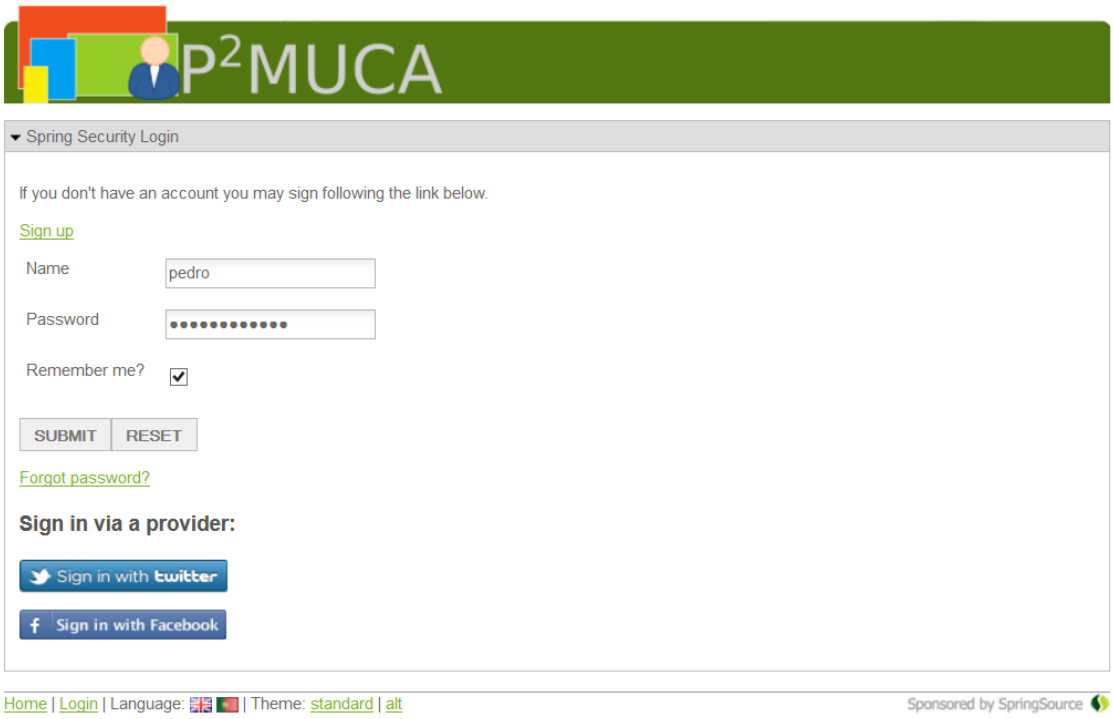


Figure 3.13: P²MUCA user login

3.6.1 Developer Options

A developer has access to the list of their current applications as seen in Figure 3.14. A new application may also be added, or an existing application edited, by using the form on Figure 3.15.

There is an additional application creation/edit mode. The form seen on Figure 3.15 only allows developers to configure the personalization model by submitting an XML configuration file [VC12]. An additional dynamic form was developed to make the instantiation of the personalization model more accessible to the users. This form is called P²MUCA Configurator, and it was initially developed as a standalone page that can be used to fully configure an application's personalization model. Resources can be dynamically added, and are made available as the building blocks of parameters. Likewise, the parameters are automatically made available to define personalization options. Figure 3.16 shows the dynamic form with some data inserted. The real form is continuous but for illustration purposes it was broken down into multiple images placed side-by-side.

Also note that the configurator offers dynamic validation of the inserted fields, e.g., a personalization option context parameter on the third column has its option marked red because the field is mandatory but is currently empty. This client-side validation ensures that the configuration generated by the configurator is always valid. However, server-side validation is still present to protect the form submit endpoint from being exploited.

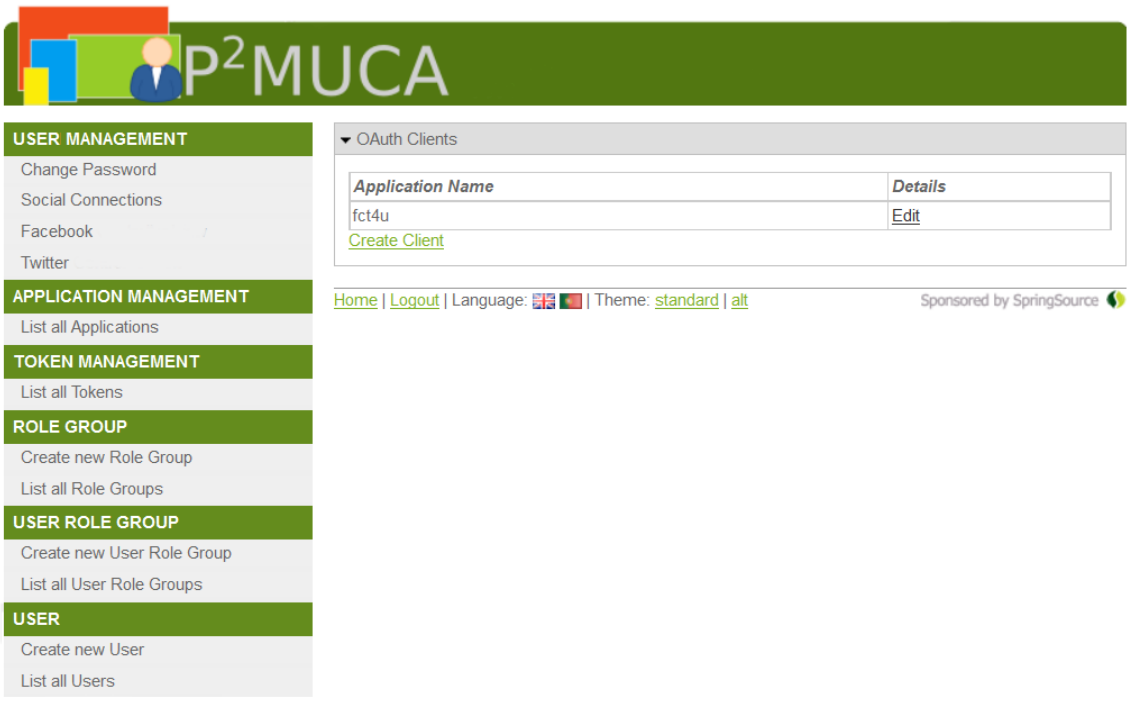


Figure 3.14: List of applications registered by a user

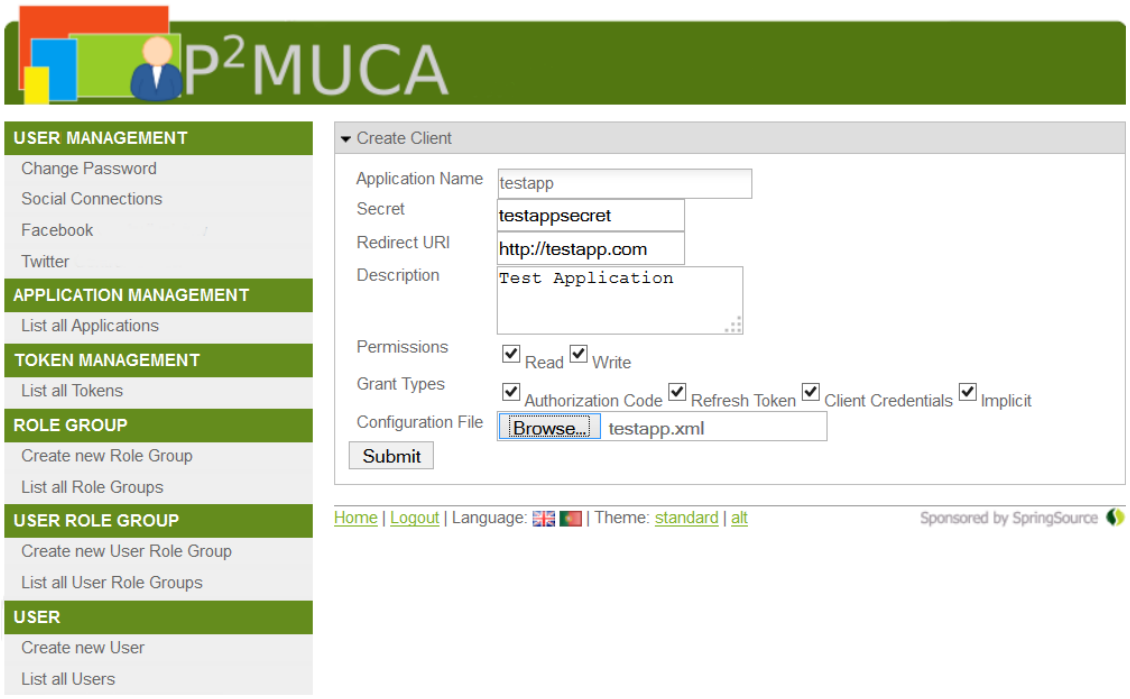


Figure 3.15: Application create/edit form

The P²MUCA Configurator has now been integrated with the website as an alternative that replaces the file upload form, since it is able to generate an equivalent XML representation that can be interpreted by CAPE as if it was an XML file. However, the file upload option is still offered because it exposes all of the available configuration options and some developers may prefer to work with an XML representation of the personalization model.

Figure 3.16: P²MUCA Configurator

3.6.2 End User Options

The platform is primarily targeted at developers, but third-party application end users also need to access some options. Besides user login and registration, end users have to be able to authorize applications that want to access their data. This is done through a page like the one seen in Figure 3.17. In the picture, the page is shown embedded into a Android application, but it can be used on any other type of application, including web-based applications.

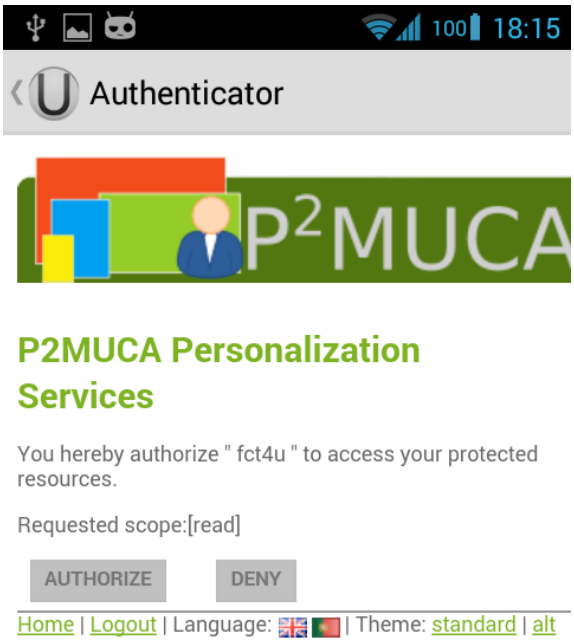


Figure 3.17: P²MUCA authorization page on an Android application

Besides giving applications the access to their data, the users should also be able to revoke that access. As such, a list of all the applications for which tokens have been issued is shown to the users. Next to each application, a delete button is available to revoke the applications' permissions, invalidating the currently issued access and refresh tokens. An example of this list can be seen in Figure 3.18.

Even after revoking access, an application may ask the user once again for permissions. It is the responsibility of the third-party developer to detect that the tokens are now invalid and that the application must repeat the authorization process.

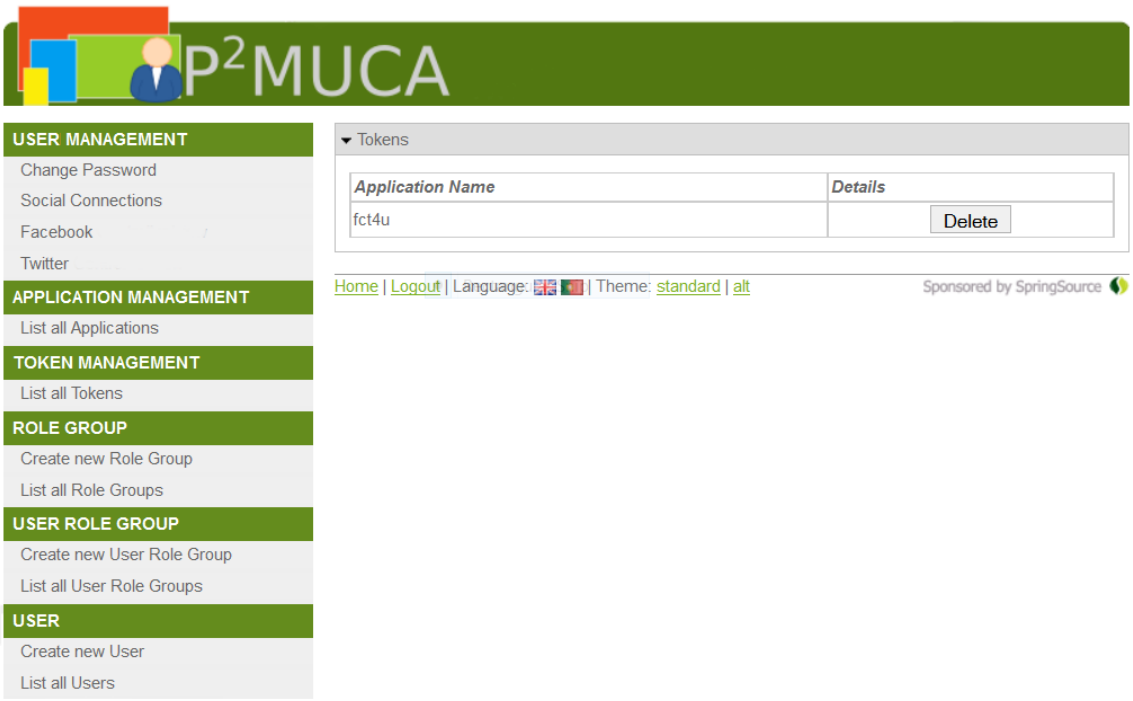


Figure 3.18: A list of user authorized applications

4

FCT4U - Interactive Public Display Application

As already noted on Chapter 1, the development of a public display application is one of this dissertation's main objectives. Such application had the objective of taking advantage of the developed personalization framework while also allowing the exploration of new interaction paradigms. With those objectives in mind, the FCT4U system was designed as a mobile and ubiquitous distributed system. The system itself is divided into two main components:

- A mobile application built for Android[1] Phones and Tablets;
- A central node responsible for aggregating all the information from the mobile devices and powering the public display.

This chapter is then dedicated to explore the development of FCT4U, focusing on design decisions, architecture, available features, the use of the personalization framework and preliminary user testing.

4.1 Design Decisions

In this section some of the more important design choices that were made to build FCT4U are presented. Those decisions cover areas such as development platforms, programming languages, techniques used for presence sensing and how asynchronous communication was achieved.

[1] Android - `http://www.android.com/`

4.1.1 Development Platforms

Android was chosen as the target platform for the mobile application, because it is currently the most successful and widely available mobile platform [Gar13]. The Android SDK[2] is free and comes with an emulator that can be used to test the application. Unlike iOS, the application can be freely installed on an Android device. In fact, the development team has multiple devices that can be used for testing.

However, as seen in Figure 4.1, due to the openness of the platform there is a high degree of fragmentation. As such, it was decided that only the more representative versions would be supported, namely Gingerbread (2.3.3 and up), Ice Cream Sandwich (4.0.3 and up), and Jelly Bean (4.1, 4.2 and 4.3). Those versions alone account for about 98% of the devices accessing the Google Play Store. Versions older than Android 2.3.3 are not supported by the application due to the extra testing effort and having to reimplement and maintain features that may have become available in later versions. Also, the tablet-only Android 3.x Honeycomb is not officially supported due to the very small market share.

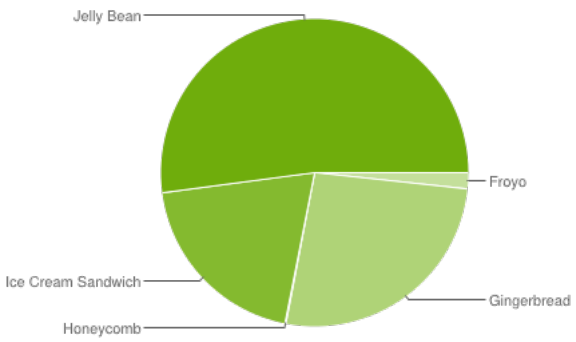


Figure 4.1: Android version distribution on Google Play on October 2013 [And13]

Besides the mobile devices, the public display itself needs to be connected to a computer running an application responsible for presenting meaningful information to the users. For that purpose a JavaFX[3] application was developed. JavaFX was chosen because it is currently the most up-to-date Java library to build graphical applications, and will eventually be an official standard part of Java [Ora13a]. Even though JavaFX could be used to build the whole user interface of the public display application, it was decided to build most of it using web technologies. This was done with the aim of reusing that part of the interface across the public display application and the companion mobile application, thus reducing the development by removing duplicate efforts in building similar user interfaces using different technologies.

That shared UI[4] was built using HTML[5], CSS[6] and JavaScript as a single web page

[2]Software Development Kit
[3]JavaFX - `http://www.oracle.com/technetwork/java/javafx/overview/index.html`
[4]User Interface
[5]HyperText Markup Language
[6]Cascading Style Sheets

that can be dynamically changed. To implement it, a library called Backbone.js [7] was used together with more common JavaScript tools, such as jQuery[8].

Also important was the use of Twitter Bootstrap[9] to help designing the web-based UI. It provides structure, component templates and responsive layout capabilities. A responsive layout is indeed very important due to the need to support the large public display application and the smaller screens of mobile devices while using the same codebase.

The web UI was then embedded in both the Android mobile application and JavaFX public display application through the use of the WebView classes present in both platforms. For bi-directional integration between the hosting application and the web UI, the JavaScript code provides an interface that can be called from the host application to perform operations to change the web UI state. Likewise, the host applications also provide an interface to be used by the web UI, so that it can call the host application to deal with certain situations (e.g., user input events). The UI provided by the mobile and public display applications is the same. However, the implementation differs due to differences between the host platforms.

4.1.2 Presence Sensing

Also important for the public display was the ability of sensing the presence of users around it. As such, Bluetooth discovery was used due to its easy availability while being relatively adequate to determine if users are relatively close or not to the public display.

A library implementing the JSR-82 standard[10] was used to interact with Bluetooth devices. This is the Java standard API to deal with Bluetooth devices. The library itself is called BlueCove[11], and even though it is no longer actively developed, it is quite possibly the only freely available JSR-82 implementation for Java SE which supports multiple Bluetooth Stacks on multiple platforms (Windows, Linux and OS X).

As of the current version of FCT4U, Bluetooth is only used to detect the presence of users around the public display site. However, through the use of the RSSI[12] value of each device, it is expected that in the future it may be possible to approximately determine how close a user is to the display and act accordingly to further improve the user experience. It is important to note that getting the RSSI value when using the Windows Bluetooth Stack is not possible. However, while using the BlueZ Bluetooth Stack on Linux it was possible to conduct some empirical tests on whether Bluetooth signal strength could accurately determine a user proximity to the public display. Those tests were not very promising and the best that seems to be possible is to determine if someone is very close to the display or if it is just passing on premises of the public display

[7] Backbone.js - `http://backbonejs.org/`
[8] jQuery - `http://jquery.com/`
[9] Twitter Bootstrap - `http://getbootstrap.com/`
[10] Java APIs for Bluetooth - `http://www.jcp.org/en/jsr/detail?id=82`
[11] Bluecove - `http://bluecove.org/`
[12] Received signal strength indication

installation. That was the main reason why the use of the RSSI value was put on hold, until a more controlled large scale experiment could be conducted, from which data could be extracted and analyzed to find patterns that could allow to infer more proximity levels. If successful, it would be possible to distinguish between actors that are really interacting with the system, spectators that are watching what is happening from afar and bystanders that are just passing on premises.

4.1.3 Communication

A crucial part of this system is the communication between the mobile devices and the public display. That communication link would have to be bi-directional to allow the public display application to push information to the mobile devices. For that purpose, multiple solutions were considered, e.g., Bluetooth connection and direct communication through TCP sockets using the campus wireless network.

The use of Bluetooth would introduce the need of pairing devices which would hinder the user experience. Other problems include the limited range and the maximum number of seven active connected devices that Bluetooth has. So the idea of using Bluetooth for communication between the mobile device and the public display was put aside.

The other obvious choice would be to use the campus wireless network to establish socket connections between the mobile devices and the public display installation. However, this would prove to be impossible because security policies of the wireless network do not allow direct connection between devices. Also, in both cases seen so far, users would not be able to communicate with the FCT4U system when they would not be in close proximity with the public installation.

To overcome these problems, the most simple idea would be to create an intermediate service that could be used as a messaging hub between the mobile devices and public display. Since this is a dedicated component for communication, it would not need to be placed in the same place as the public display application, and would be easier to find somewhere to host it in a publicly accessible network.

This communication hub component was developed as a Java Servlet that can be added to any Servlet container, thus making it easy to deploy. Given that Servlets are mostly used to develop HTTP-based application and with the aim of using a standard protocol the idea of using WebSockets arose. The WebSocket protocol specification (IETF RFC 6455 [Fet+11]) has been finalized in 2011, while the a JavaScript API is still being standardized by W3C [Hic11]. It offers a bi-directional communication channel over a TCP connection. Its main objective is to bring full-duplex communication to client side applications running on web browsers; however, it can be used in any other type of applications as long as there is a library that conforms to the protocol specification.

The WebSocket Protocol is an independent TCP-based protocol. It is only related to HTTP because the initial request to open a WebSocket connection is made through

HTTP by issuing an upgrade request. The protocol can also run on top of TLS, providing a secure end-to-end connection. The protocol is currently supported by multiple web servers/servlet containers, such as Tomcat, Jetty, JBoss and Glassfish.

So far, the WebSocket protocol covers all of the requirements needed to build the communication hub needed for FCT4U and, as an added bonus, it allows the direct interaction from web-based applications if the browser supports WebSockets. In fact, the only reason why the WebSockets' connections have to be handled by the host applications is because the WebKit[13] versions embedded with JavaFX and Android do not support WebSockets. So, to connect to the WebSocket communication hub, Jetty[14] was used for the public display application and Autobahn Android[15] for the mobile application.

Since WebSocket support in multiple Servlet containers is still not fully standardized, the use of a framework that abstracts the underlying server implementation is very useful to allow the deployment of the component on different containers. One of those frameworks is Atmosphere[16], which provides tools to build WebSocket-based applications, while offering multiple fallback mechanisms (e.g., HTTP long-polling and streaming, Comet, etc.). It was chosen due to the quality of its documentation and examples, and also because it was one of the most complete free options for Java.

4.2 System Architecture

After discussing some important design decisions, it is time to present the overall architecture of the system. A high-level architectural overview can be seen in Figure 4.2.

The architecture is based on a series of mobile devices' users interacting with a centralized engine placed on the Public Display installation. Thanks to the bi-directional link provided by WebSockets the user input, preferences and other gathered data can be passed in real-time to the systems' engine.

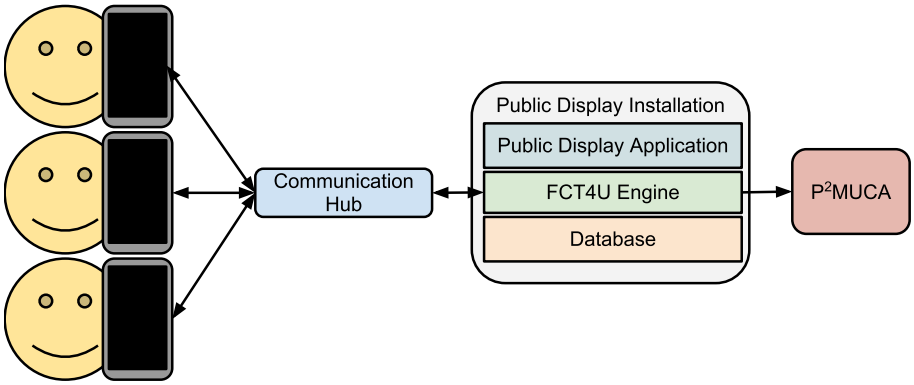


Figure 4.2: FCT4U architectural overview

[13] WebKit is a layout engine software component designed to allow web browsers and other applications to render web pages - `http://www.webkit.org/`

[14] Jetty HTTP Server and Servlet Container - `http://www.eclipse.org/jetty/`

[15] Autobahn Android WebSocket library - `http://autobahn.ws/android`

[16] Atmosphere Realtime Client Server Framework - https://github.com/Atmosphere/atmosphere

On the other hand, the engine feeds the Public Display Application with the pre-processed and pre-selected information according to the user profile built from user's preferences, interaction data history, and context. Building such user profiles is assisted by the use of P^{2}MUCA. Those profiles are also fed into the mobile application to show appropriate information to the user on his private smaller screen, while keeping the ability to control the experience on the larger public display.

The engine is currently running in the same application as the Public Display Application itself, but it is decoupled from the presentation layer and could be separated from it to power multiple displays if the need arises. In such scenario, multiple **Public Display Application** components would interact with a single **FCT4U Engine**. That engine is backed by a relational database, with EclipseLink[17] being used as the JPA[18] implementation together with a MySQL database[19].

It is also important to note that all communications going through the **Communication Hub** are encoded in JSON[20] format for easy parsing and interpretation by all parties involved. The JSON library used in Java was Jackson[21]. Besides reading and writing simple JSON data, Jackson has very useful object serialization features. For example, there is a *UserProfile* class that contains multiple information about the user, including nested information in other classes. Jackson is smart enough to understand common Java conventions and easily convert an in-memory UserProfile object to a JSON representation that can be sent as text over a WebSocket connection and be converted back to an in-memory object at the receiving end. This architecture is extremely useful to build interactive applications based on mobile devices serving as the entry point of a system that needs near realtime bi-directional communication. Therefore, the same architectural principles could be used by other applications with similar requirements.

It is true that for this project some of the communication could be done using more traditional methods, such as HTTP requests. However, the use of WebSockets encompasses the need of continuously streaming user interaction without having to make an HTTP request per interaction, thus improving system responsiveness and performance.

At the same time, it offers the much needed feature of having data pushed to the device as soon as it becomes available, without the need of polling the server. Older techniques (e.g., long-polling and HTTP streaming) are inherently unidirectional (server to client) and an additional connection must be opened from the client to the server every time the client needs to send information.

[17] EclipseLink - `http://www.eclipse.org/eclipselink/`
[18] Java Persistence API
[19] MySQL Relation Database System - `http://www.mysql.com/`
[20] JavaScript Object Notation - `http://json.org/`
[21] Jackson - `https://github.com/FasterXML/jackson`

4.3 Available Features

The purpose of FCT4U is to offer useful information to users in a large screen, while allowing them to control what they are seeing with a mobile device that also displays similar information on a smaller display. Considering that the large screen is a public display, some sensitive information may only be shown on the mobile device's screen, or it can be shown on the large screen only if the system determines that the user is alone or among friends.

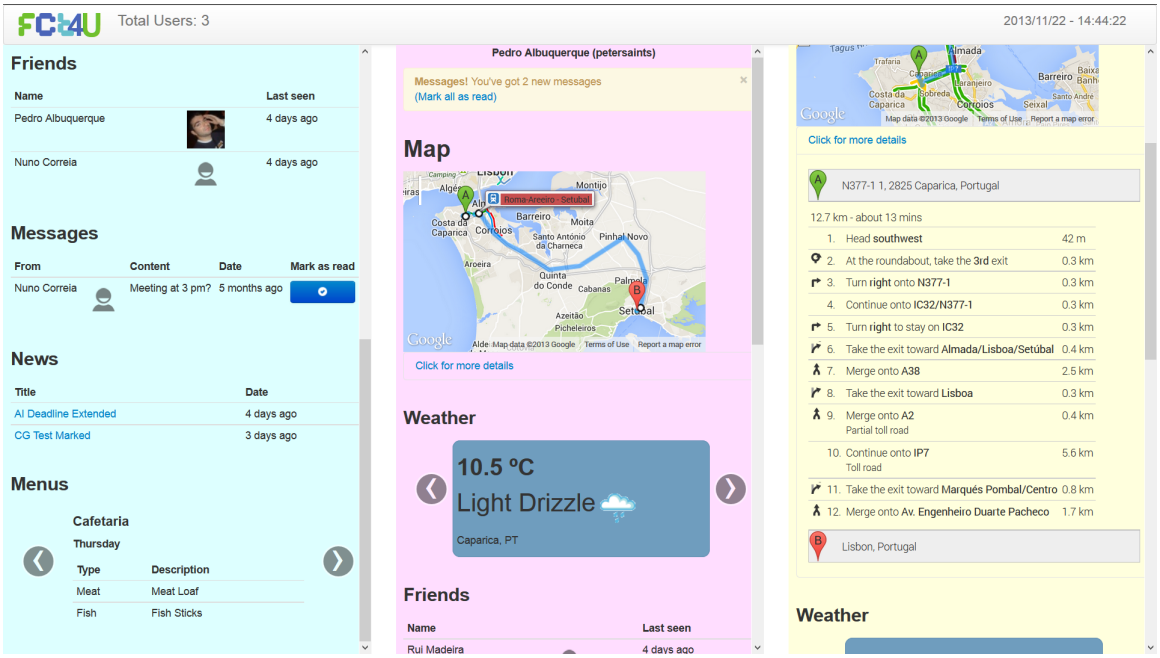


Figure 4.3: FCT4U information sample

A sample of all the information that can appear on a public display for three users is shown in Figure 4.3. On the mobile device of a single user the information shown would be simply one of the columns of Figure 4.3. An example of such scenario can be seen in Figure 4.4.

While interacting with the mobile view on the user's device the public display is updated in realtime, thus allowing the user to scroll and change the information shown. This also works the other way around, if the public display application is running on a touchscreen enabled computer, such as the one used for tests in the lab.

4.3.1 Widgets

Each piece of information on FCT4U is programmed as a widget for a specific type of information that can be shown to the user. There are currently six widgets available:

- Map widget;

- Weather widget;
- Lunch Menus widget;
- News widget;
- Messages widget;
- Friends widget.

Each widget was developed as a Backbone.js view to encapsulate its layout and logic. This way widgets can be dynamically added as needed. Therefore, widgets can be easily added and modified without changing the rest of the web UI. The six widgets currently available are presented in this subsection.

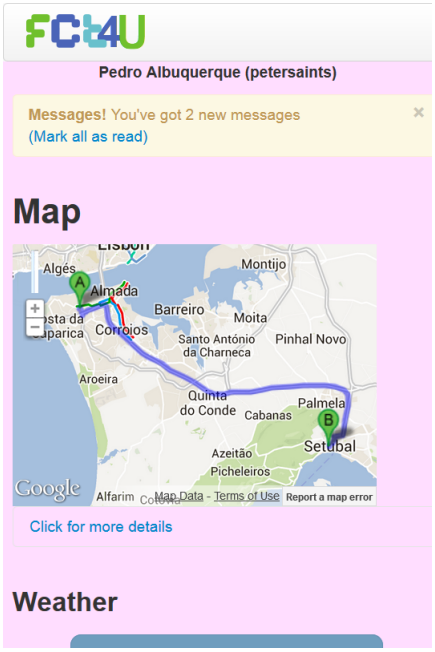


Figure 4.4: FCT4U mobile view sample

4.3.1.1 Map

The Map widget has the objective of giving the user an overview of where s/he is and her/his home. The home was selected as a possible place where the user may want to go next, but any other place could be used instead if available and the need arises. Additionally, it will show directions between the two locations along with traffic information, if the user uses a private transport, or public transport information if the user uses the bus or train to move around. All the information, i.e., the map itself, routes, traffic and public transport information, is obtained through the **Google Maps JavaScript API**[22].

[22]Google Maps JavaScript API v3 - `https://developers.google.com/maps/documentation/javascript/`

Both scenarios can be seen on Figure 4.5. The user on the right uses public transports, while the user on the left uses private transport. If a user clicks *Click for more information*, the widget shoes detailed route information.

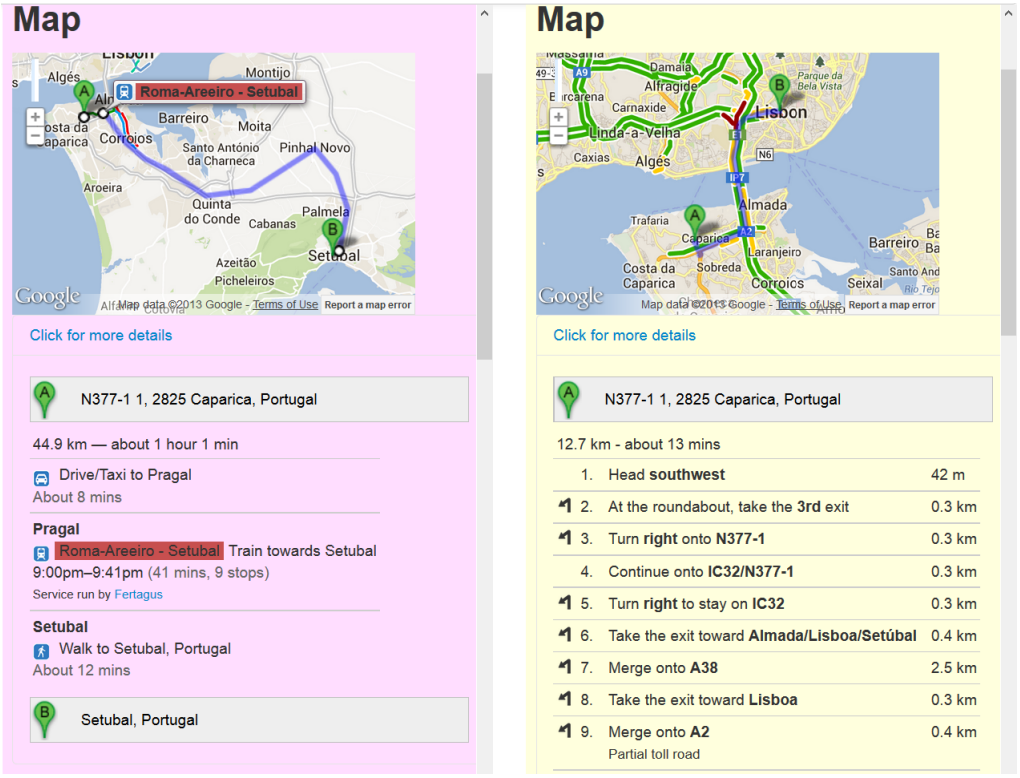


Figure 4.5: Map widget modes

4.3.1.2 Weather

The weather widget simply shows the current weather, and the forecast for the following days, in places that may be of interest to the user. This widget can be seen on Figure 4.6.

Figure 4.6: Weather widget

The places shown are her/his home location, work place and the campus itself. If any of those places are very close together, then one of them will be disregarded and not be shown. To move through the multiple places the user should swipe her/his fingers over

the widget or just use the arrows.

The weather provider is Open Weather Map[23] because it was the only free provider that could be found which offered a virtually unlimited amount of API calls. All the other providers had free limits that were too low for the current application.

4.3.1.3 Lunch Menus

The Lunch Menus widget was built to show information about the menu for the current day on the multiple cafeterias and restaurants that exist on campus. Currently, the provided information is fictional, because there is no feed or service to get that information.

Nevertheless, the widget can be seen in Figure 4.7. Similarly to the weather widget, the users can move between different bars and restaurants by swiping over the widget or using the arrow buttons.

Figure 4.7: Lunch menus widget

4.3.1.4 News

The News widget can show news that may interest the user. Currently it only shows the information from the faculty's website news feed[24]. However, in the future, the widget may present more personalized information selected and mixed from a large array of sources.

Without any kind of interaction the widget looks like Figure 4.8. It shows the news title and how much time has passed since it was published. However, if someone clicks on the news title (whether directly on a touch screen or through the mobile application), it will show a QR Code on the public display. This QR Code can be used by someone to follow a link to the news source, thus allowing users to share news with other people around them. This scenario can be seen on Figure 4.9.

On the mobile devices screen, if a user clicks on the news title, a small overlay tooltip will appear with a link which can be clicked to go directly to the news source on the mobile device (see Figure 4.10). This happens because this is the action used to tell the

[23]Open Weather Map - `http://openweathermap.org/`
[24]FCT/UNL News Feed - `http://www.fct.unl.pt/noticias/rss.xml`

public display to show the QR Code, so a second click on the tooltip is needed to really open the news on the user's device.

Figure 4.8: News widget

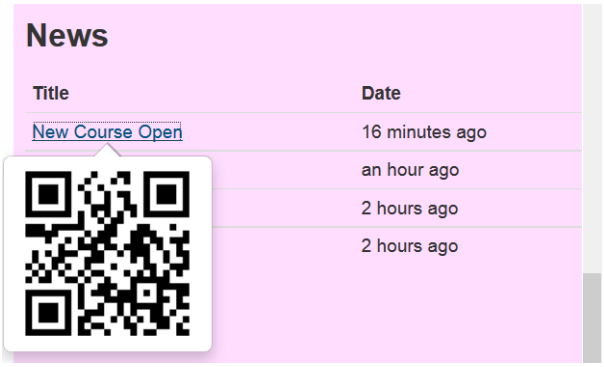


Figure 4.9: News widget showing a QR Code

Figure 4.10: News widget showing a link to the news source

4.3.1.5 Messages

The Messages widget shows personal messages sent by other users. To send a message, the mobile application must be used to select the person the user wants to send the message to and to write the message content.

Within FCT4U a friend is someone that uses the system and is a Facebook friend. To determine someone's friends, that person must first associate hers/his Facebook account to FCT4U through the mobile application. Then, anyone who is a user's Facebook friend and has done the same association will be considered a friend. Those friends will then appear in the list where the user chooses to whom s/he wants to send the message to.

The messages are limited to 140 characters just like Twitter. This prevents users from spamming very long and hard to read messages that do not fit well into the FCT4U user interface layout.

Sent messages will be shown to the recipient the next time s/he passes near the public display or requests an information refresh through the mobile application. The message is shown in a widget like the one in Figure 4.11. When available, the Facebook profile picture is shown next to the name of the sender and the blue button lets the users mark a message as read.

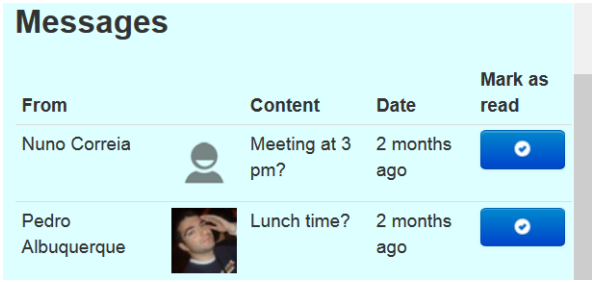


Figure 4.11: Messages widget showing a few messages

Besides the widget itself, a yellow alert bar is shown on the top of each user column with the number of unread messages and a link that lets the users mark all messages as read (see 4.4).

4.3.1.6 Friends

Still related to friends is the possibility of knowing whether some of user's friends have passed recently near the public display. That is the objective of the Friends widget, which shows the name of a user's friends, their Facebook's profile picture and how much time has passed since they have been seen. This widget can be seen in Figure 4.12.

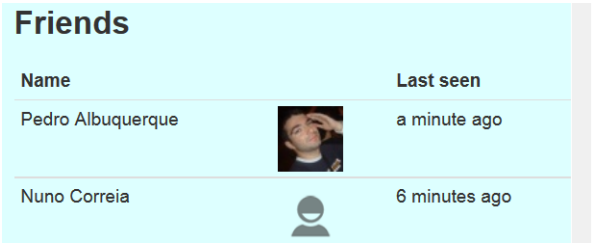


Figure 4.12: Friends widget

Just like the Messages widget, friends are determined using Facebook integration to determine which users are friends of another user. This is done through the use Facebook Graph API[25] and FQL[26] with the help of a Java library called RestFB[27].

[25]Facebook Graph API - `https://developers.facebook.com/docs/reference/api/`
[26]Facebook Query Language - `https://developers.facebook.com/docs/reference/fql/`
[27]RestFB - `http://restfb.com/`

Among other things, those APIs allow FCT4U to gather information about friends and whether they have authorized the FCT4U application, thus determining if some friend is also an FCT4U user because each user using Facebook integration has her/his Facebook ID stored on FCT4U's main database. That way the widget is filled by searching the current day's detection history for all of the user's Facebook friends that use the system.

4.3.2 Mobile Application

A mobile application for Android 2.3.3 and up is a key feature of FCT4U. Its most important tasks are user registration and detection. The user registration process allows the application to identify each user of the system and to start targeting content for that specific user. Since the application is using P^{2}MUCA, its own user authentication system is not needed. So FCT4U uses P^{2}MUCA credentials as their own through OAuth 2.0 with an extended token validation endpoint to use OAuth, not only for authorization, but also for authentication. Since P^{2}MUCA also supports signing up and logging in using Facebook and Twitter, those providers can also be used with FCT4U. The user login or registration page running on a smartphone can be seen in Figure 4.13.

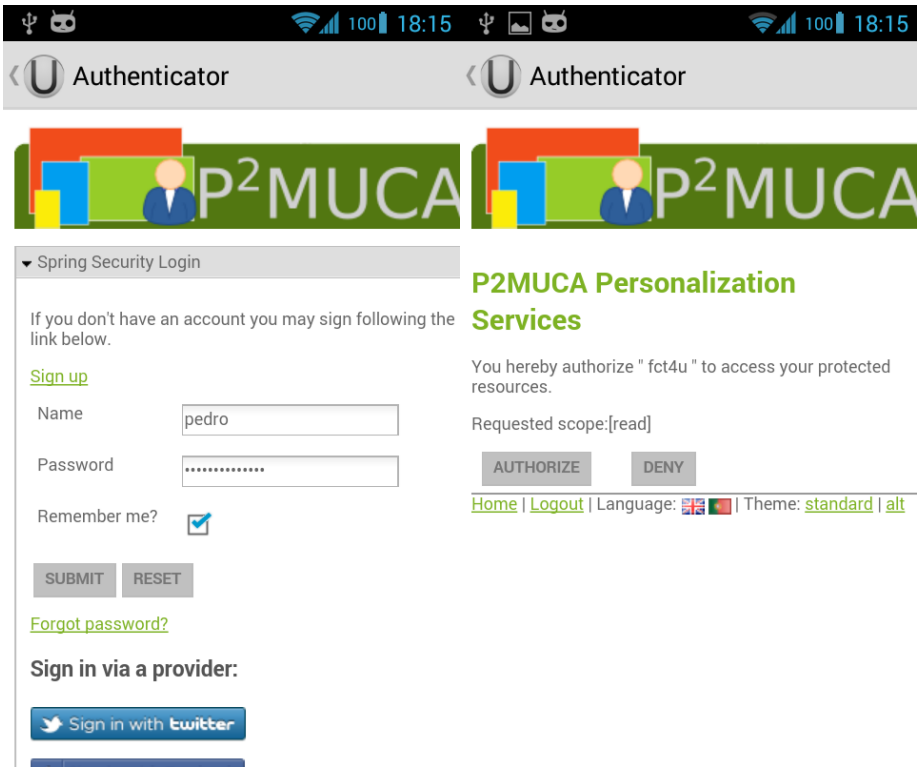


Figure 4.13: FCT4U Authenticator. Signing in on the left and authorizing on the right.

Coupled with the user registration is the detection of nearby devices. When a user goes through the authentication process, the application also gathers the Bluetooth device unique address and associates it to the just authenticated user. This allows the Bluetooth discovery routine running on the public display application to identify to which user a

device belongs to.

The identification that a device belonging to a user is nearby is used to show relevant information to that user. So, even for a completely passive mode, the public display still needs the mobile application to help with the association between users and devices. Even if the application did not offer any more features, this association would have to be made.

However, the mobile application offers more features. It allows the user to set some preferences about the mobile application itself and influence the information shown to the user. It is also through those preferences that a user can associate its Facebook account to FCT4U (see Figure 4.14), thus allowing the Friends and Messages widgets to work. The Facebook association also enables that all of the user's Facebook friends that also use the application appear as possible message recipients on the send message screen (see Figure 4.15).

The application also enhances the features already available through the public display itself. The same information that is available on the public display can also be viewed on the mobile device even if the user is not near it. It can also be used as an input device to interact with the public display. If a user is detected near the public display, all interactions done on the mobile device are replicated to the public display. In fact, this replication of actions between private and public displays is supposed to be the main user input of the FCT4U system, as seen in Figure 4.16.

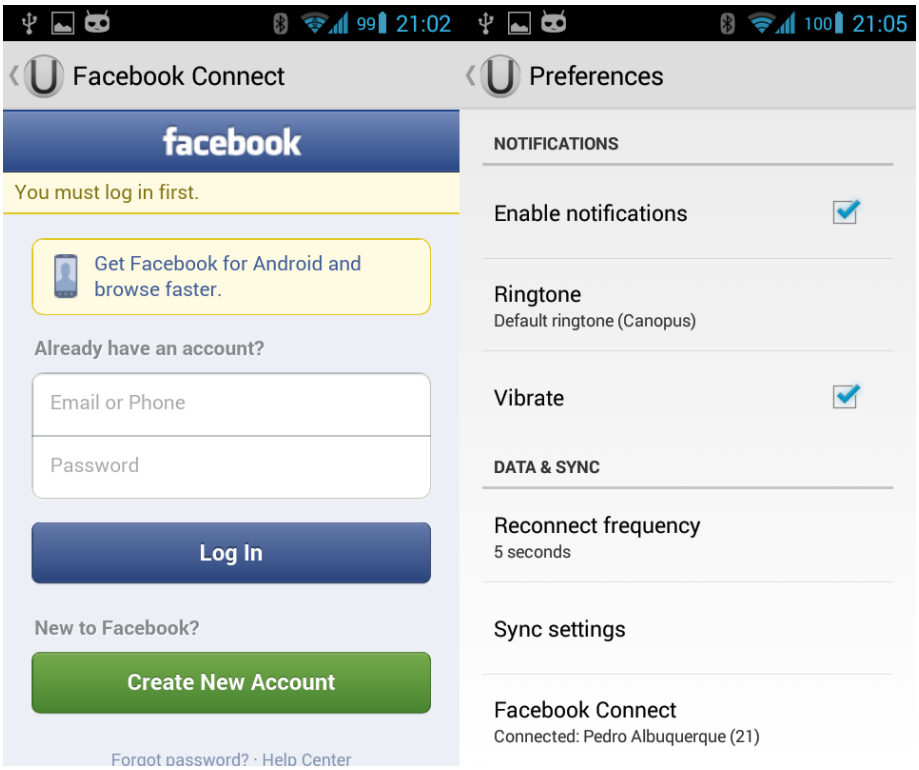


Figure 4.14: FCT4U Facebook integration. Facebook sign in on the left and some of the extracted information on the bottom right.

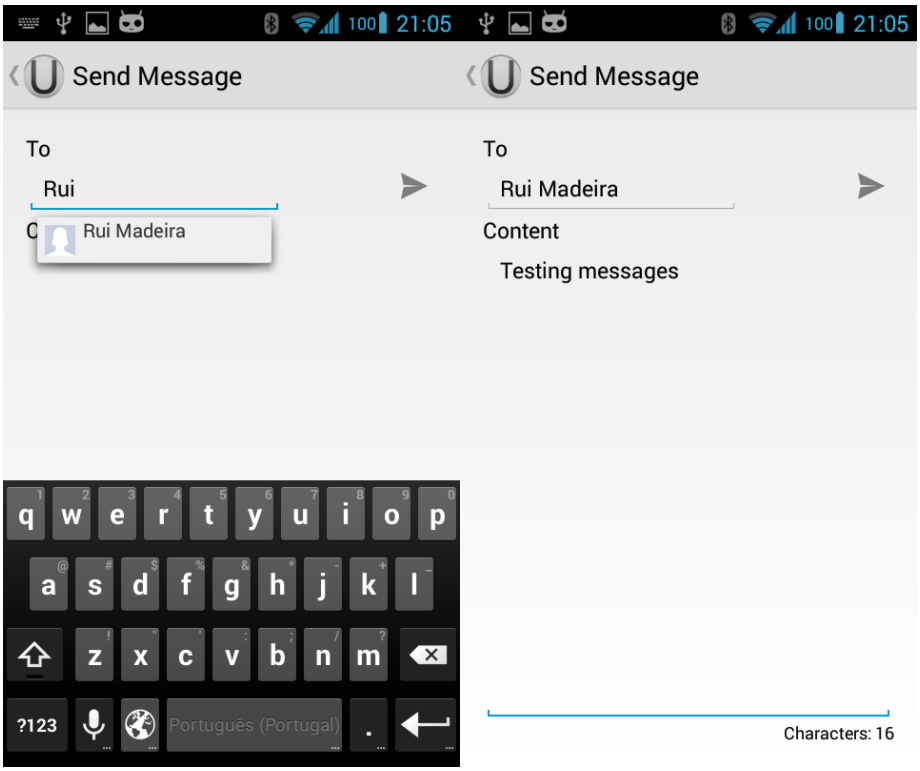


Figure 4.15: Sending message using the mobile application. Recipient auto-completion on the left and full interface on the right.

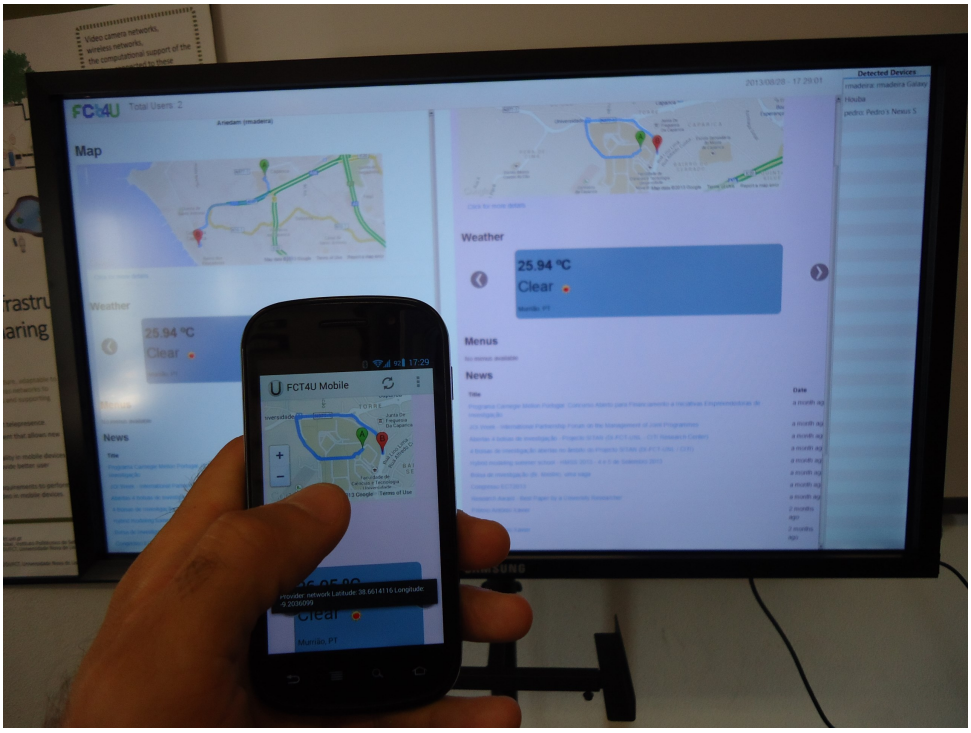


Figure 4.16: Someone using FCT4U with a smartphone

The other available option is direct touch interaction on the public display. This option is only realistic if a large touch-screen is available and/or it is desirable to have the public display at arms reach. It is better suited for semi-public display installations, where the number of people around the display is usually more limited and there is a low risk of damaging the more expensive touch-screen. The mobile application is usually preferred for fully public approaches.

Thus, the mobile application is a core component of the FCT4U system. In fact, the system could not work without it. Even if an alternative registration and detection method was thought of, it would probably be not as good from a user experience point of view, given the current state of technology. So, if FCT4U is to eventually become a production ready system, the application will have to be ported to support other popular mobile operating systems, starting with iOS, which is the second most used mobile OS.

4.3.3 Automatic Place Identification

Some of the available widgets use location data to provide more useful information. The map widget shows the way home from the faculty campus, the weather widget shows weather information for the campus, home and work, and the lunch menus widget that shows information about meal places on campus.

All these locations can be simply set from the user preferences screen available on the mobile application. However, while trying to make the application smarter and more dynamic, it came the idea of continuously collecting the user location to automatically extract those locations.

A background service was implemented in the Android application that tries to get the user location every minute. If the user has GPS enabled, and it is possible to get a fix in a reasonable amount of time, then GPS is used as the location source. Otherwise, the less accurate wireless LAN and cell tower location service bundled with Android is used. GPS technology, wireless LAN and cell tower location based services were already discussed in 2.4.3.

So, the location service running on the user's mobile device continuously determines its location and sends that information through a WebSocket connection to the public display installation, where the location is stored on the database for further processing.

That post-processing takes place whenever a user is detected by the public display or requests a refresh of information through the mobile application. On such occasions, the public display engine will create a user profile with all the information to be shown to the user. While doing that, if the user has not explicitly set some of his favorite locations, the system will try to guess what those locations are based on the stored location history.

It is for determining those locations that the density-based clustering algorithms discussed in 2.4.4 are used. The one used was DBSCAN because an implementation was readily available for use as part of the Weka[28] library, a Java library for machine-learning

[28]Weka (Waikato Environment for Knowledge Analysis) - `http://www.cs.waikato.ac.nz/~ml/`

used by the previously developed CAPE.

It is true that DBSCAN is not the ultimate density-based clustering due to a tendency to generate false positives, even though it has a good detection rate of the expected true positives. However, other algorithms are also not better in all cases, so it is difficult to select which algorithm will have the best results for the FCT4U use case. The only algorithm that seemed to be very good all around, according to [Bha09], is DPCluster. However, due to the time limitations, the fact that it is a complex algorithm, and because DBSCAN provides good enough results, it was decided not to implement it. In fact, considering that DBSCAN major problem are false positives and FCT4U is usually only interested in the cluster with the highest number of points, it is very unlikely that such cluster is a false positive. Those will most likely be some clusters that are caught while moving between meaningful places, so they should have a low number of points.

The value of 5 was chosen for $MinPts$ because location history is recorded every minute. So, if a user is not staying in the same area for at least about five minutes, it probably means that the user is moving between different places. For the ϵ parameter 15 was chosen because the distance is being calculated in meters and that radius distance around a point seems to be a good compromise between disregarding very close places as different places and considering different places the same one.

For all of the three locations that the system is trying to determine, only the last week of location history is being considered. For determining the home location only the data recorded between 0h and 8h is used, because most people will probably be at home at that time. The mean point of the largest cluster is considered to be the home location. Similarly, the mean point of the largest cluster between 9h and 18h is used as the work location, due to encompassing most common work hours. For the lunch place the same is done for the 12h to 14h interval.

The results for home and work location look promising. A good estimate of where a person lives and passes regular work hours is achieved. Even if it does not guess it with a very high accuracy, anything within a few hundred meters is more than adequate for the map and weather widget.

However, the lunch place is more problematic. There are some bars and restaurants in the same building and due to some variability on the location fixes it becomes nearly impossible to distinguish between them. Besides that, those lunch places sometimes overlap because of being placed on different floors of the same building or because of being very adjacent places. Nevertheless, for different buildings this approach works well enough. So that information can be used to select the closest meal place to the largest cluster center and show it as the default on the widget. Other smaller clusters can also be used to order the remaining possible places.

It may be possible to attenuate this problem by tweaking the clustering algorithm parameters, using a better algorithm, or even creating a custom algorithm mixing concepts of existing ones that deals better with this scenario. However, that will probably not be

weka/

enough to solve the problem altogether, because looking at the raw data, as seen in Figure 4.17, clearly shows that it is difficult to determine where a user has been within the same building.

This problem comes from technical limitations imposed by the Android's Wi-Fi location service, which lacks the needed accuracy and precision for this particular task. Additionally, GPS is not a better option because most locations are indoors, or in very close proximity to buildings. This means that, even when a location fix is possible, it usually has very poor precision due to interferences. The only solution would be developing a dedicated campus location service.

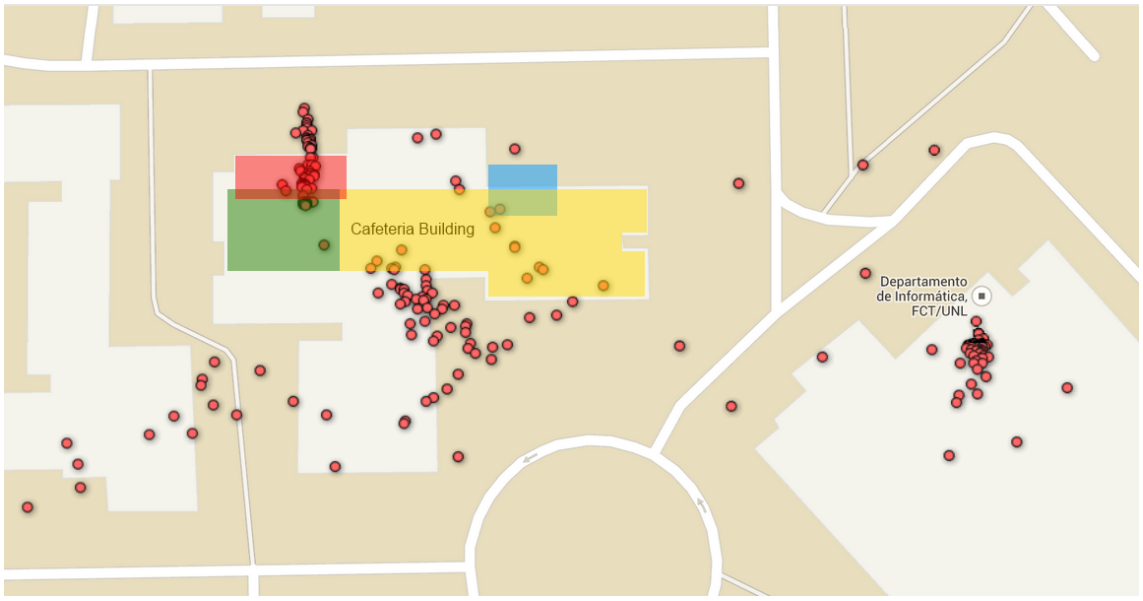


Figure 4.17: Location data on a partial map of the campus. The colored rectangles represent different bars and restaurants on the same building.

Besides trying to guess where someone lives, works and lunches, the location data is also used in the attempt of automatically determining whether someone uses public transports. To do that, the number of times that a user has been close to the major public transport hub near the campus is counted. If it passes a certain threshold and the user has no explicit preference of the transport type, than it is assumed that lately s/he has been using public transportation.

This is just yet another attempt to make the application smarter. It may not be the best approach if someone happens to often go to the public transportation hub area for another reason. But overall, it is a valid assumption that if someone is detected in that area a certain number of times a week that s/he is using public transports.

4.4 Personalization Applied to FCT4U

Testing and validating the P^{2}MUCA platform was one of the main reasons why FCT4U was developed. Therefore, the used personalization model must be instantiated, i.e., the underlying CAPE data model must be configured to accommodate the new application.

That can be done through the P^{2}MUCA website, either by submitting an XML configuration file or by using the provided user interface to make the configuration easier.

The basic data sources of the model are the resources, which store numerical values related to the user interaction, preferences and options. On FCT4U, some of those resources count the number of times a user uses the application or interacts with a given widget, how much scrolling is done, how often the user is detected by the public display or how many times a week the user is nearby the major public transport hub in the campus area.

All of those resources are then used as the basis to create personalization parameters. Those parameters take algebraic expressions, giving different weights and establishing relationships between each resource. For example, the *communicative* parameter is configured as seen in (4.1). While trying to determine if someone is communicative, or not, that expression is used to calculate a value which will be used to separate users into one of the categories.

$$0.2 * messagesRead + 0.3 * messagesReceived + 0.5 * messagesSent \tag{4.1}$$

Multiple parameters are combined to make personalizations, which consist of multiple possible combinations of the categories defined by the parameters. For example, a personalization called *viewOrder* may have an option called *messagesFirst* that is chosen when someone is simultaneously *communicative* and does not use the lunch menus widget too much (i.e., a *notMenuAholic*). In such case, this personalization may be used to tell the FCT4U engine to place the messages widget on the top of the user column view.

Additionally, the personalization model allows context segmentation. So the resource values that power the whole personalization process can be associated with the current context of a user. In such case, context specific values of the resources are used instead of the global ones. This enables FCT4U to have different behaviors depending on the user context.

For example, in the *viewOrder* it makes sense to segment personalization depending on whether the current user's time is before or after lunch. The values that influence whether to show lunch menus first will most probably be related to the time of day. So, maintaining and using a set of values for the morning and another for afternoon and night, allows the personalization to be more dynamic and context-aware.

The remaining subsections will present and detail all of the used resources, parameters and personalizations.

4.4.1 Resources

Table 4.1 lists all of the defined resources, including their names, type and purpose.

Name	**Resource Type**	**Description**
detectionCount	App Data (*increment*)	Counts the number of times a user is detected near the public display
uses	App Data Resource (*increment*)	Counts the number of times a user launches the mobile application
interactionCount	App Data (*increment*)	Counts the number of times a user interacts with any widget
messagesSent	App Data (*increment*)	Counts the number of messages sent by a user
messagesReceived	App Data (*increment*)	Counts the number of messages received by a user
messagesRead	App Data (*sum*)	Counts the number messages read by a user
newsClicked	App Data (*increment*)	Counts the number of news clicked by a user
newsViewed	App Data (*increment*)	Counts the number of news viewed by a user
weatherInteractions	App Data (*increment*)	Counts the number of interactions with the Weather widget
menuInteractions	App Data (*increment*)	Counts the number of interactions with the Lunch Menus widget
mapInteractions	App Data (*increment*)	Counts the number of interactions with the Map widget
mapDetailInteractions	App Data (*increment*)	Counts the number of times that the Map widget details are toggled
nearPublicTransportHubs	App Data (*sum*)	Counts the number of times that a user is detected near public transport hubs
lunchInterval	Context (*hour interval*)	Indicates whether a user current time is before (8h to 14h) or after lunch (14h to 8h)
company	Context (*value*)	Indicates whether a user current time is *alone*, *accompanied* or *accompanied only with friends*
userType	Preference	It defines the type of user (*student* or *teacher*)
transportType	Preference	It defines the preferred transport type (*private* or *public*)

Table 4.1: Personalization resources used by FCT4U

4.4.2 Parameters

Table 4.2 lists all of the defined parameters based on the previously presented resources.

Name	Formula	Options	Description
userActiveness	$0.5 * uses + 0.3 * interactionCount + 0.2 * detectionCount$	low; medium; high	Measures how active a user is
transportProfile	$0.7 * transportType + 0.2 * nearPublicTransportHubs + 0.1 * userType$	private; public	Used to separate users that use public transportation or not
userTypeProfile	$0.9 * userType + 0.1 * transportType$	student; teacher	Used to distinguish user types
communicative	$0.2 * messagesRead + 0.3 * messagesReceived + 0.5 * messagesSent$	communicative; notCommunicative	Used to determine how much someone uses the short messaging features
menusAholic	$1.0 * menuInteractions$	menusAholic; notMenusAholic	Used to determine how much someone uses the Lunch Menus widget
newsAholic	$0.7 * newsViewed + 0.3 * newsClicked$	newsAholic; notNewsAholic	Used to determine how much someone uses the News widget
weatherAholic	$1.0 * weatherInteractions$	weatherAholic; notWeatherAholic	Used to determine how much someone uses the Weather widget
mapAholic	$0.6 * mapInteractions + 0.4 * mapDetailInteractions$	mapAholic; notMapAholic	Used to determine how much someone uses the Map widget

Table 4.2: Personalization parameters used by FCT4U

4.4.3 Personalizations

Multiple personalizations were implemented based on the previously defined resources and parameters. Those personalizations are used to determine when to adapt the FCT4U to certain situations.

4.4.3.1 *mapMode* Personalization

The first personalization is called *mapMode*. This personalization is used to determine what kind of information to show on the Map widget. If a user uses a private transport, i.e., a car, it should show traffic information. On the other hand, if a user uses public transports, information about its schedules and routes should be shown.

The options of this personalization are summarized in Table 4.3.

Name	Parameters	Description
publicTransport	transportProfile → *private*	When selected show route between current place and home by car, along with traffic information
privateTransport	transportProfile → *public*	When selected show route between current place and home by using public transports, along with relevant public transport schedules

Table 4.3: *mapMode* personalization options

4.4.3.2 *privacyMode* Personalization

The next personalization is called privacyMode. This one is used to determine if the application should be more careful with the level of detail of the information shown by default in the Map widget. It complements the *mapMode* personalization by adding into account the *userType* resource. This is important, because for a professor it may be undesirable to show its home location fully zoomed in by default, so to avoid students knowing exactly where a teacher lives.

Additionally, this personalization uses another type of resource not used so far. If passed in the API call, it uses a context resource to further inform the application if a user is accompanied with strangers, friends or alone when using the application. A user's company is determined based on other devices detected nearby and whom they belong to. Since Facebook integration is already being used for some widgets, it is also used to classify if the detected devices (at the same moment) belong to a friend or not.

The multiple options defined for this personalization are summarized in Table A.1 of Appendix A. Those options cover all possible combinations of the used parameters.

4.4.3.3 *greetingsMode* Personalization

The following used personalization is the greetingsMode. This one is simply used to distinguish between different types of users in order to greet them in different ways when the applications detects them. It does not power any useful feature but it may be useful to give users some feedback about their activities. The personalization options are based on the *userTypeProfile* and *userActiveness* parameters and are presented in Table 4.4.

Name	Parameters	Description
inactiveStudent	userTypeProfile → *student*; userActiveness → *low*	A student that does not use the system very much
inactiveTeacher	userTypeProfile → *teacher*; userActiveness → *low*	A teacher that does not use the system very much
averageStudent	userTypeProfile → *student*; userActiveness → *medium*	A student that uses the system moderately
averageTeacher	userTypeProfile → *teacher*; userActiveness → *medium*	A teacher that uses the system moderately
activeStudent	userTypeProfile → *student*; userActiveness → *high*	A student that uses the system intensively
activeTeacher	userTypeProfile → *teacher*; userActiveness → *high*	A teacher that uses the system intensively

Table 4.4: *greetingsMode* personalization options

4.4.3.4 *viewOrder* Personalization

The last and more important personalization is the one regarding the order of the widgets shown in FCT4U. This personalization takes into account metrics regarding the use of interactive widgets. Additionally, the *company* resource is taken into account in options that have the *communicative* parameter set to *communicative*, so that the system can decide if it should show messages on the public display, due to their private nature. In order to take account of cases in which the *company* context is not available, options without it are also present.

Also, since this personalization is mostly based on user interaction habits, this personalization uses context segmentation. It takes into account resources that are specific to each *lunchTime* context hour interval with a weight of 25% when calculating the parameters values. This way, if someone uses the Lunch Menus widget a lot during the morning, but not so much during the afternoon, the system is able to put the person into different usage profiles depending on the time of day.

All of the possible combinations between the used parameters are presented in Table A.2 of Appendix A. The name of each option is representative of that combination and it is interpreted by FCT4U to reorder the widgets shown on the web UI.

4.5 User Study

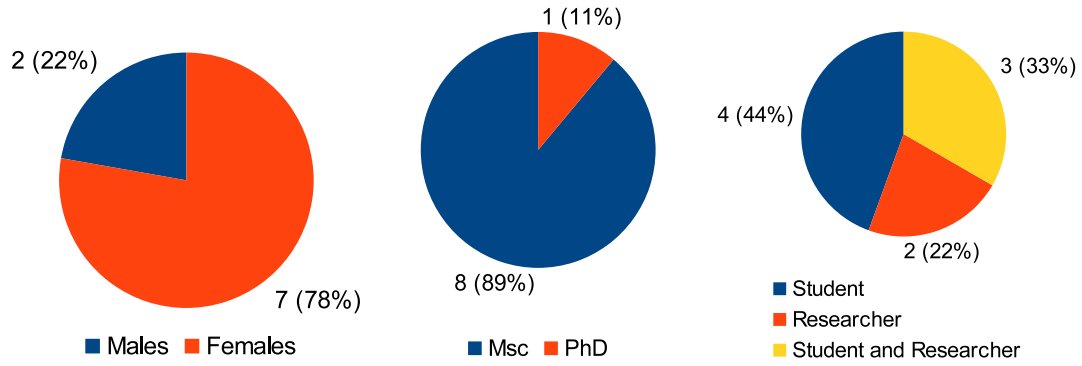
A first usability evaluation was conducted, in which participants described their thoughts, observations, and actions as they interacted with the first FCT4U prototype. The study was concerned with the evaluation of FCT4U concept, if users believe this is a useful system in terms of supporting and helping their daily lives in the faculty campus. The interaction modes and the pieces of information made available were the main focus. Questionnaires were favored over personal interviews, although the latter appeared in an informal way while observing the testing process. Each user used the system for about 30 minutes and were free to explore it once the test was over.

The results are a starting point, since a second user study is planned for future work in which the same users will participate along with others. In that phase the main focus will be dealing with personalization, thus a longer test is required to assess how users react to it over time.

4.5.1 Participants and Design

Participants were recruited from the CITI research center, in the FCT/UNL campus, since the experiment was conducted at CITI's IMG laboratory. A 46 inches touchscreen was used for the public display application and multiple Android smartphones and tablets were used to run the mobile application. A photography of a user using a smartphone to interact with the public display can be seen in Figure 4.16.

A total of 9 subjects took part in the user study, aged from 26 to 39 years, with an average age of 30.5. 2 of the subjects were female, as seen in 4.18(a). All subjects had a master's degree, except one who was a PhD researcher (Figure 4.18(b)) and all of them were either students or researchers at the department (Figure 4.18(c)).

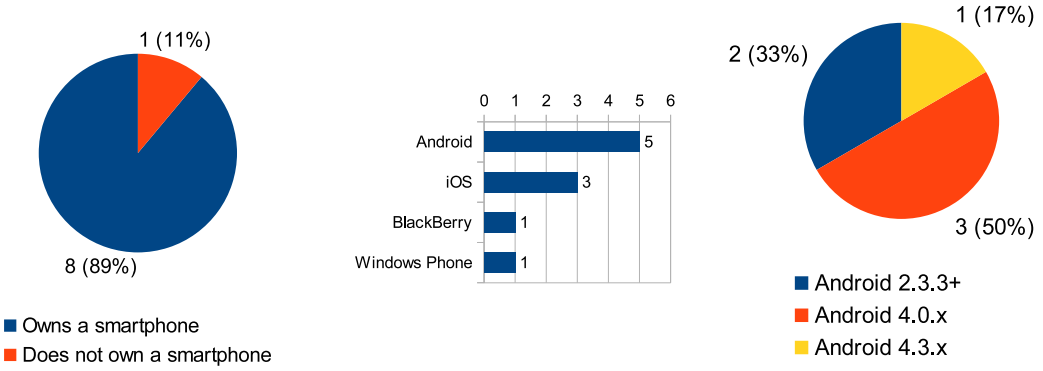


(a) Gender (b) Degree (c) Role

Figure 4.18: About the test subjects

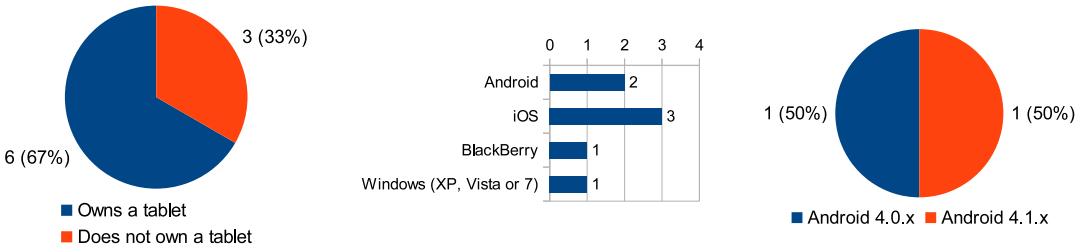
All but one user had at least one smartphone, in which 5 of them had an Android-based smartphone (Figures 4.19(a) and 4.19(b)). Tablets are not as common as smartphones since their massification is still ongoing, so only 3 of the users said that they have

one and, of those, 2 have an Android tablet (Figures 4.20(a) and 4.20(b)). All of the Android devices owned by the test subjects ran Android 2.3.3 or higher, so almost 70% of the participants could use the current system in their daily living. According to the statistics, developing the mobile application for Android was the best choice to support as many users as possible with a single platform



(a) Smartphone owners (b) Smartphone operating system (c) Android versions

Figure 4.19: About users' smartphones



(a) Tablet owners (b) Tablet operating system (c) Android versions

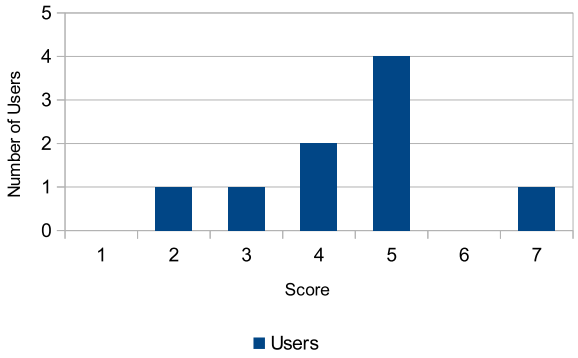
Figure 4.20: About users' tablets

After the initial briefing, testing was broken down into three major tasks, in which users used different interaction scenarios in order to study how they reacted to them. The three scenarios were:

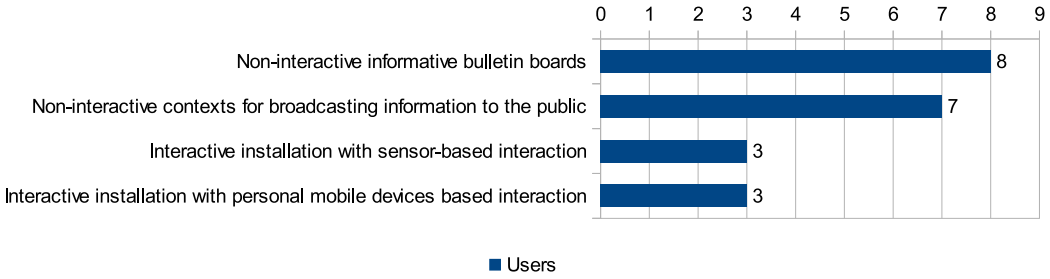
- Passive interaction by just being detected by the system and seeing information in the public display
- Active interaction using a mobile device running the FCT4U's Android application
- Active interaction using the touch-screen modality of the public display

After finishing each task, participants were asked to fill in an anonymous questionnaire. All questions that were asked to rate something used a 7-point Likert style scale.

One thing that became apparent from talking to users and from the questionnaire results, is that they are already used to have contact with static displays that provide no way for people to interact with, but a few already had contact with interactive installations. This is expected since the users come from a research group focused on interactive and multimedia applications. For an overview of the answers about how much experience users have with this type of system, see Figure 4.21.



(a) Experience with public displays



(b) Number of users that have used some types of public display installations

Figure 4.21: Test subjects' experience with computers and public display applications

4.5.2 Results and Discussion

Based on the results from those questionnaires, a summary of the main findings grouped by task is presented.

4.5.2.1 Task 1 - Passive Interaction Mode

Most participants (7 out of 9) gave a positive score to the system usefulness, with a median of 6. Also, in general, the users considered that the system has the potential of saving them time, especially if the widget order and information is tailored for their needs. The majority of the participants (7 out of 9) considered that the time it took the system to detect their presence was well above adequate. However, in general the users considered

that the information that the passive approach provided was just above adequate with a median of 5. The results from which these claims were drawn can be seen in Figure 4.22.

From this first scenario, in which users stood still in front of the display once they were detected, it was possible to conclude that without interaction mechanisms only a small portion of the available information could be seen, mainly the map, weather and lunch menus widgets which are the ones shown first by default. This is corroborated by the results presented in Figure 4.23.

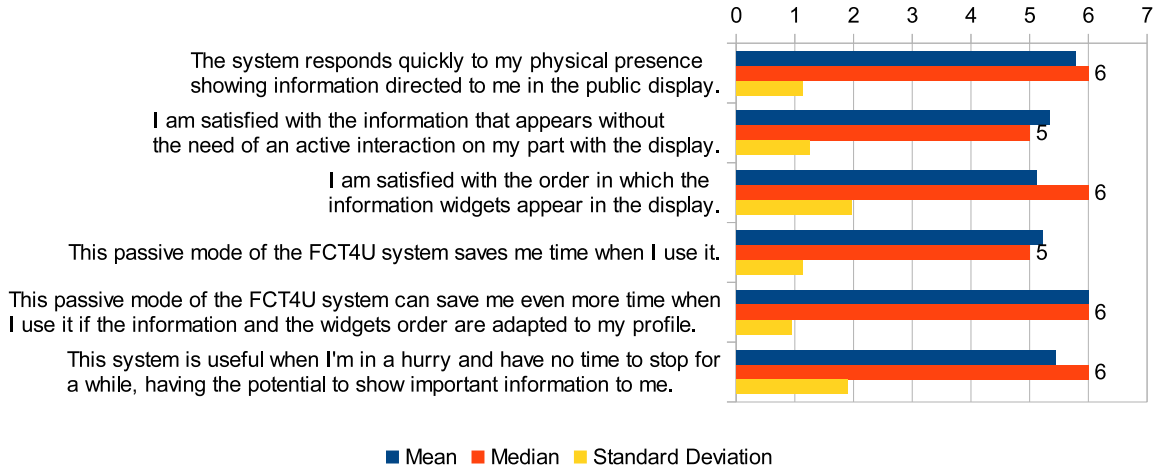


Figure 4.22: Results after the first task

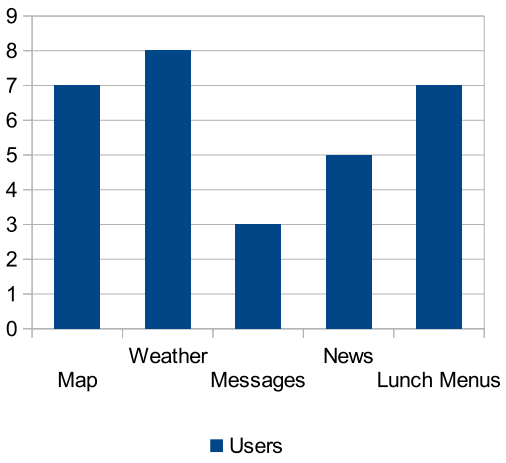


Figure 4.23: Question: *What are the pieces of information you did find in the public display?*

4.5.2.2 Task 2 - Mobile Application Interaction Mode

The second scenario involved using a mobile device to interact with the public display while exploring the mobile application. The vast majority considered that the mobile application is useful and enriches the overall experience, with the questions regarding those topics getting medians of 6 and 7 (Figure 4.24).

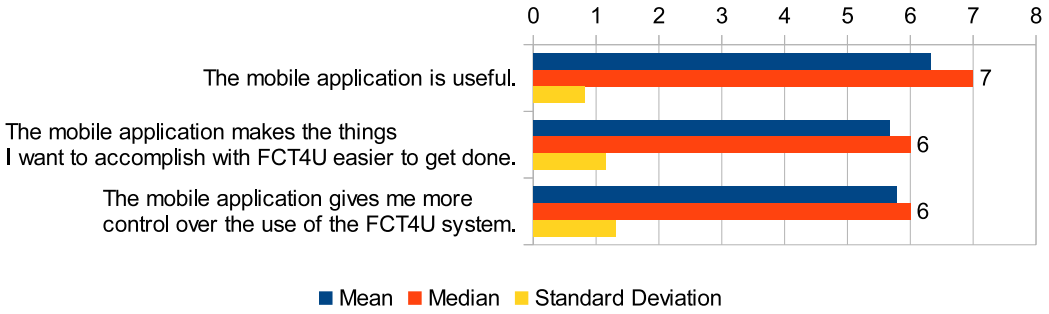


Figure 4.24: Questions regarding usefulness and control provided by the mobile application

In terms of ease of use, user effort, recovering from mistakes and how the application works together with the public display, the scores were a bit lower, with medians around 5 and 6 (Figure 4.25). Nevertheless, higher grades (medians of 6 and 7) were given when it came to consider the application pleasant to use and quick to master (Figure 4.26).

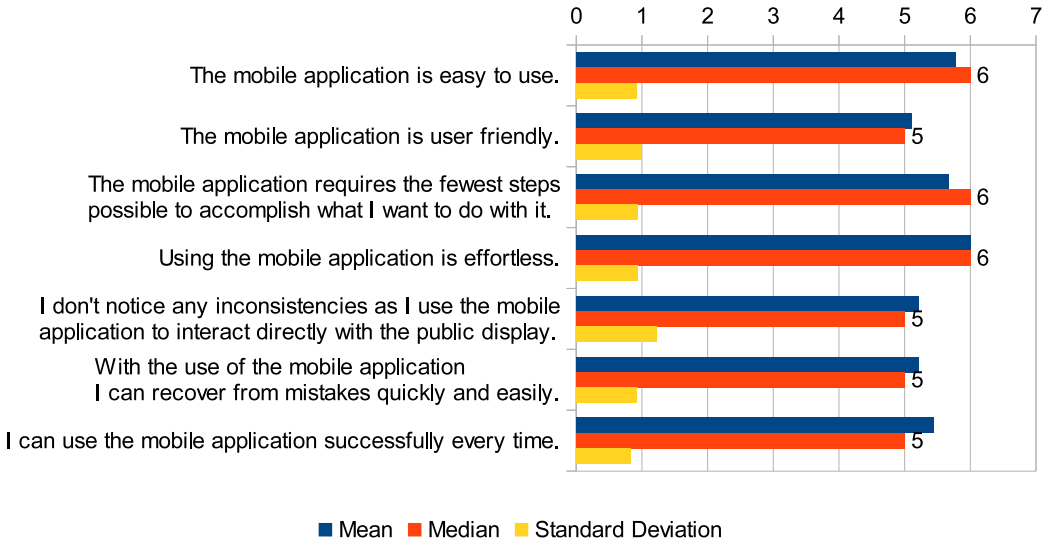


Figure 4.25: Questions regarding the mobile application's usability and its relationship with the public display

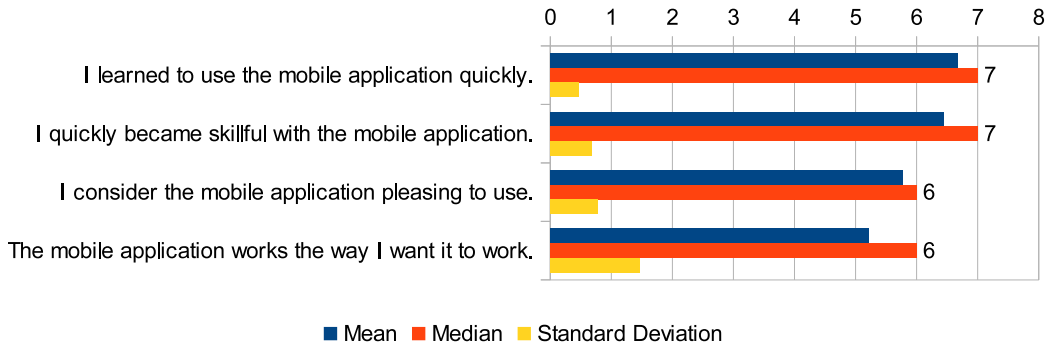


Figure 4.26: Questions about how pleasant and easy to master the mobile application is

In fact, according to the first question of Figure 4.27, all of the participants considered this interaction mode at least moderately appropriate to the FCT4U system, with a mean of 5.88, while there were no negative responses to the second question.

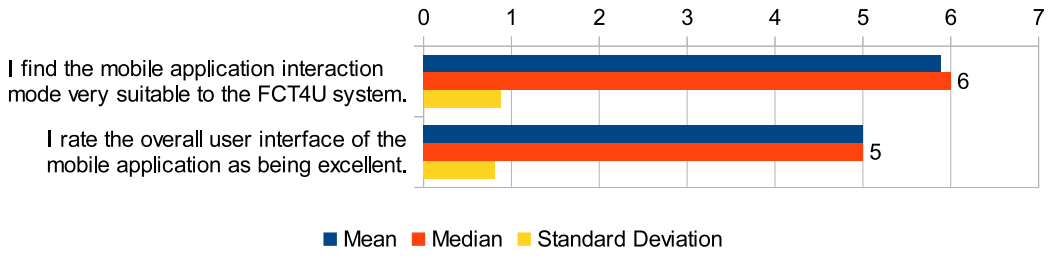


Figure 4.27: Questions about how appropriate and how good is the mobile application

The results regarding the mobile specific feature of sending short messages can be seen in Figure 4.28. Some users did not considered it very useful giving negative grades (3 out of 9), while others considered barely useful (3 out of 9), but the remaining 3 considered it very useful. So, as can be seen, the opinions diverge when it comes to this feature. The problem may be that, given so many other communication alternatives, users do not see the true potential of leaving messages in the public display to be seen by others upon arrival.

Moreover, as the last question in Figure 4.28 suggests, users do not consider the system as being a collaborative tool, although some recognize its potential. Therefore, future work will focus on making messaging and collaboration features more useful and refined, e.g., allowing to send group messages, mobile notifications when messages are received and expanding collaboration beyond the News widget QR code by allowing to share other pieces of information.

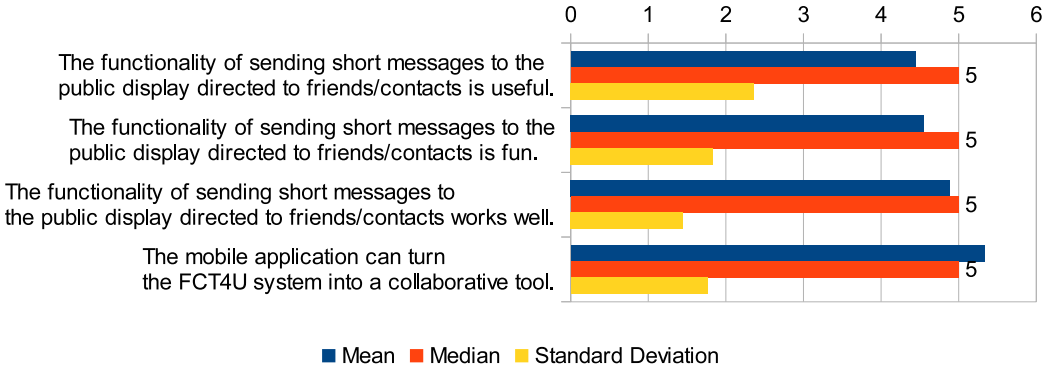


Figure 4.28: Questions regarding the sending messages feature and if the system can be used as a collaborative tool

4.5.2.3 Task 3 - Touch-screen Interaction Mode

Since a large touch-screen was available for testing, a final and third scenario was explored, in which users were encouraged to try out the system by touching the screen.

The acceptance of the touch-screen was a little bit lower than that of the mobile application, since the scores for usefulness and the degree of control are a bit lower than with the mobile application, as seen in Figure 4.29. This may be due to the fact that most people are not used to interact with large touch-screens in an up-right position, thus not providing the same feedback that holding a smartphone or tablet does. Also, since they had already used the mobile application, they have expressed that they see little advantage on having the touch mode as well, thus corroborating our initial design choice of focusing on the development of a companion mobile application, instead of relying on the system running on a touchscreen. Nevertheless, the majority of users (7 out of 9) considered that the touch mode was easy. Some users expressed the concern that this interaction mode may not be very comfortable, specially if others are also using the touch-screen at the same time.

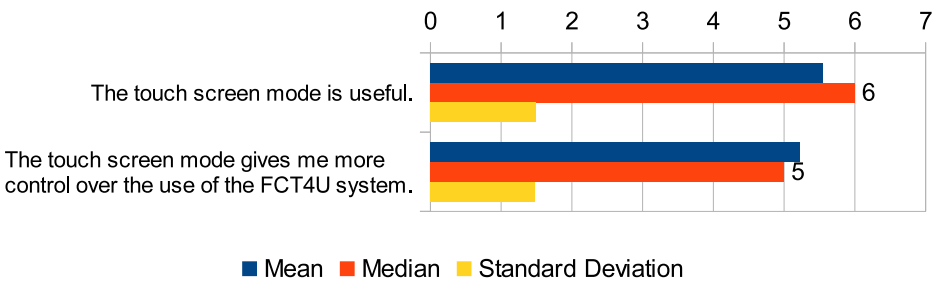


Figure 4.29: Questions regarding the usefulness and the degree of control offered by touch-screen interaction

Additionally, Figures 4.30 to 4.35 show more questions regarding the touch-screen interaction. Since these questions focused more on the scenario of having multiple users using the touch-screen simultaneously, it is harder to analyze these results. However, in general the results were positive, with medians always equal or superior to 4. In fact, considering only when the user is alone, the medians are all 6 or 7. Medians vary between 5 and 6 when 2 users are present using the touch-screen and it is only when 3 users are present that some medians fall to 4. The comparison between the different number of users using the system is further explored in 4.5.2.4.

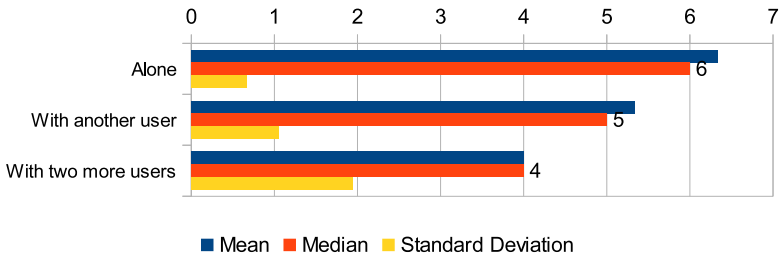


Figure 4.30: Question: *I find the touch screen interaction mode suitable to the goal of FCT4U when I interact with the display*

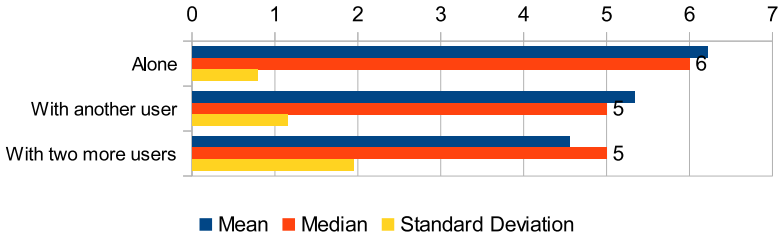


Figure 4.31: Question: *The user interface works well with the touch screen mode when I interact with the display*

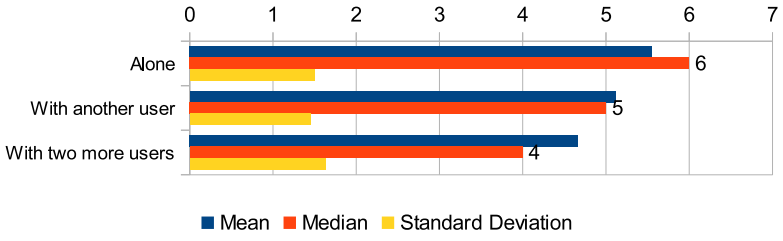


Figure 4.32: Question: *I don't notice any inconsistencies as I use the touch screen to interact with the FCT4U system when I interact with it*

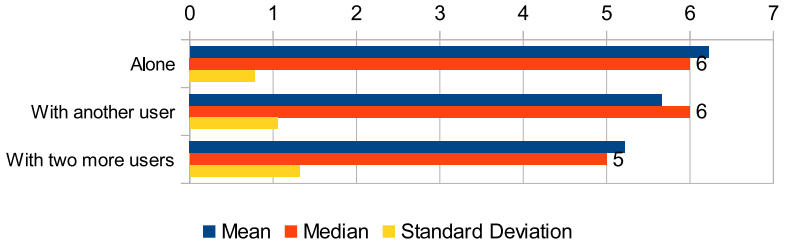


Figure 4.33: Question: *I can use the touch screen mode successfully every time I interact with it*

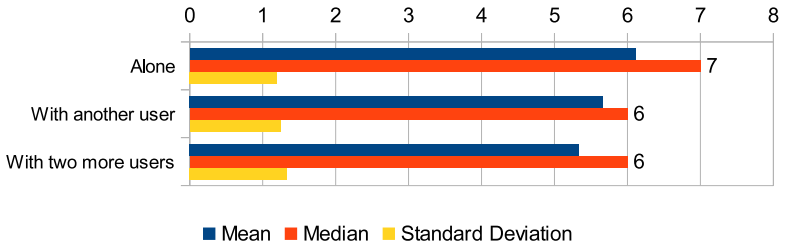


Figure 4.34: Question: *I learned to use the touch screen mode quickly when I interact with the display*

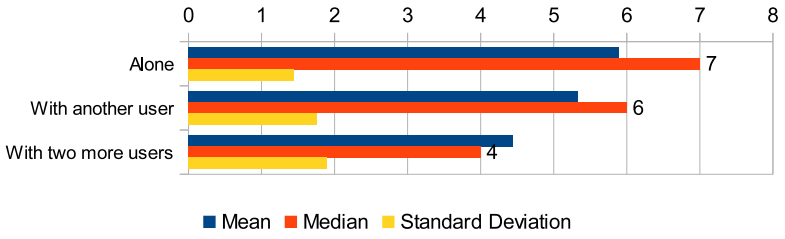


Figure 4.35: Question: *I consider the touch screen mode pleasing to use when I interact with the display*

4.5.2.4 Discussion and Comparison of Interaction Modes

Regarding the split of the screen to accommodate multiple users, it was established that people usually prefer to have the display only for themselves in any of the scenarios. However, they may cope with the need of having to split the screen to share the system with others, due to the fact that they gave the same scores, or slightly lower, as the number of users increases. This happens with the touch-screen interaction mode, as seen in Figures 4.30 to 4.35, but also with the passive mode, as can be seen in Figure 4.36.

Finally, the remaining results about the FCT4U as a whole can be seen in Figure 4.37. Overall, the system was rated with a mean of 5.33 as pleasant to use, having no negative scores, but most (6 out of 9) with either neutral or just slightly positive scores. Only 3 out of 9 gave high scores. So, this show us that while the system receives positive scores, it still has some room for improvement. Other gathered metrics indicate largely the same,

with a special highlight for the ease of use and confidence which stand out among the others.

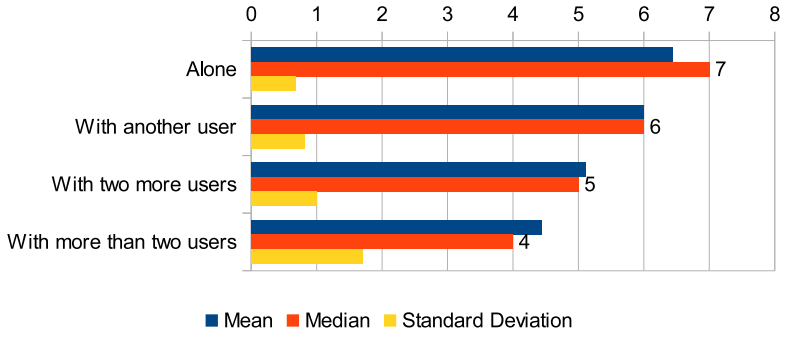


Figure 4.36: Question: *I can read and understand the information when I am in front of the display*

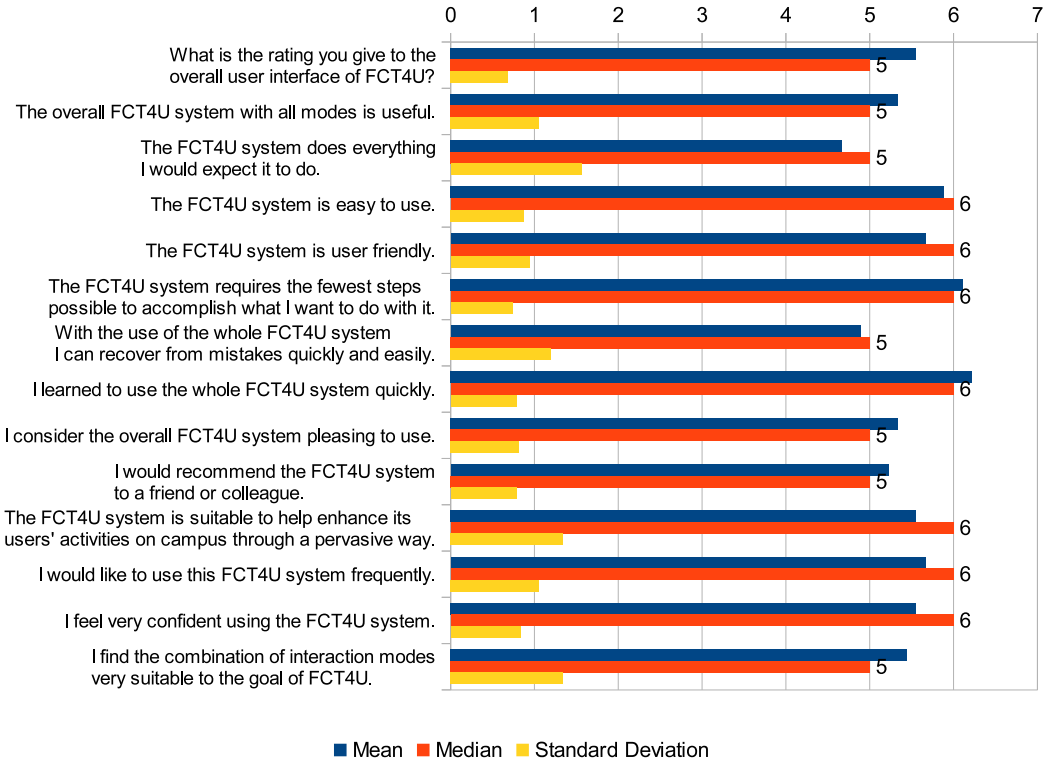
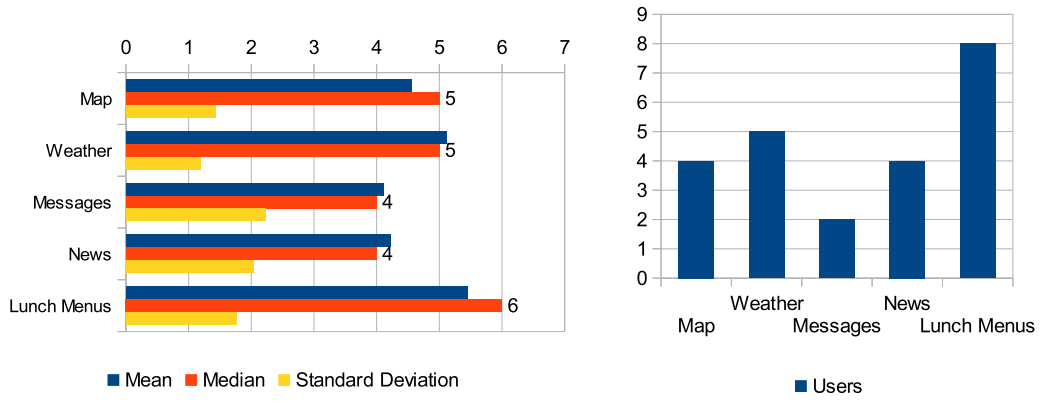


Figure 4.37: Responses about the overall FCT4U system

The individual widgets had similar scores to the overall system's scores (see Figure 4.38(a)), but the Messages and News widgets were the ones with the lower mean scores while also having the greatest variability, thus reiterating that those are the ones that need

more future work to make their unique sharing and collaboration features really useful.

On the polar opposite is the Lunch Menus widget, which is the only one to have a median score of 6, thus showing that the participants are very interested in knowing which place is the most indicated one to eat. These conclusions are also corroborated by Figure 4.38(b), which shows that the Lunch Menus widget was considered a favorite by 8 out of 9 users, while the Messages and News widgets were only considered favorites by 2 and 4 users, respectively.



(a) Rating of each widget

(b) Number of users that have chosen each widget as one of their favorites

Figure 4.38: Comparing widgets

Unexpectedly, when asked directly, the users rated their user experience with the 3 different scenarios as quite similar (see Figure 4.39). Despite the results being very close together, the mean is actually higher for the passive mode, then comes the mobile application and the lowest mean is given to the touchscreen.

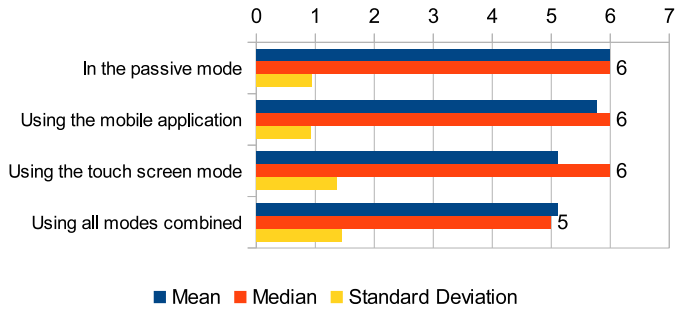


Figure 4.39: Question: *The user interaction with FCT4U is effective*

These results contradict the scores for the usefulness of mobile and touch-screen interaction, with means of 6.33 and 5.55 respectively, and conversations we had with most users, where the mobile application seemed to be the preferred interaction mode.

However, from another point of view, those results are not that strange, because most users (7 out of 9) said that they would use the passive mode (see Figure 4.40). This confirms that personalization is important in this kind of systems. If the passive mode shows more relevant information to each user, than the satisfaction can only increase. In fact, most users had a positive opinion about that proposal when asked directly.

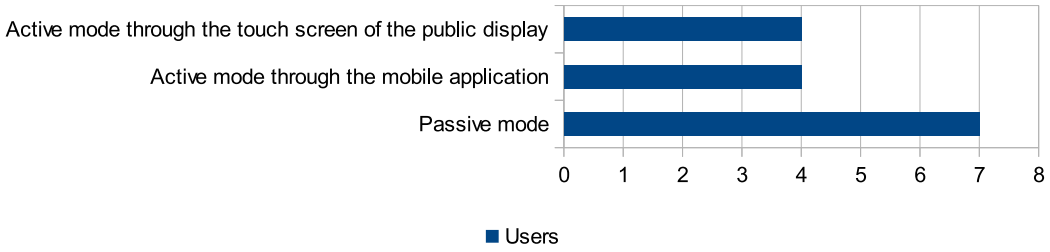
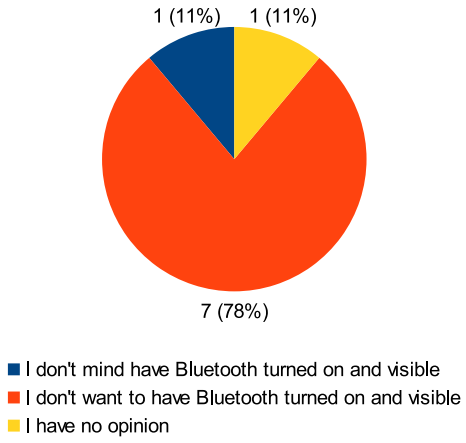
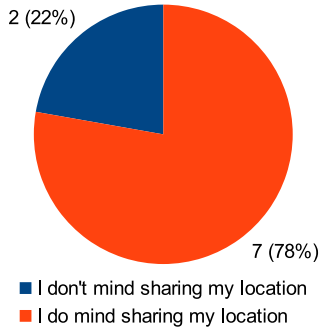


Figure 4.40: Number of users saying that would use each interaction mode

In terms of privacy concerns, 7 out of 9 users did not want to have Bluetooth turned on and visible, while one did not mind and another had no opinion, as seen in Figure 4.41(a). This might be a problem in terms of adoption, because the system depends on it to detect users. However, this barrier can certainly be overcome since users seem generally interested in using the system and in recommending to others, as can be seen by looking at the relevant questions in Figure 4.37.



(a) Opinions about Bluetooth visibility



(b) Opinions about location tracking

Figure 4.41: Opinions about privacy

Aside from increased battery drain, there is no big downside on having Bluetooth turned on. The privacy impact is minimal. The Bluetooth device address hardly serves to identify anyone unless the person has associated it with a system such FCT4U, in which the identification is in fact intended. The friendly name can be set to something

that does not personally identify someone, so others around will not pose a privacy threat just because a person has his/her device visible.

Something similar happened when asked about the location tracking, i.e., 7 out of 9 (but not the same participants as before) did not want to share where they continuously are so that the system could learn their favorite places. This is not a major concern because those places can still be set manually and the location tracking can be turned off.

Some user apprehension was already expected and this feature was mainly developed as a research challenge. Users seemed to like the idea of having the Lunch Menus widget personalized according to their habits (see Figure 4.42), and location tracking is required for that feature at least during the lunch time. So, an option may be added to the system, in which users allow location tracking only during those times if they really value a more personalized experience when using the Lunch Menus widget.

In fact, as results in Figure 4.42 suggest, users want control over what the application shows on the display, so more fine grained location tracking options, besides just on and off, and other relevant privacy options can be added to improve trust in the application.

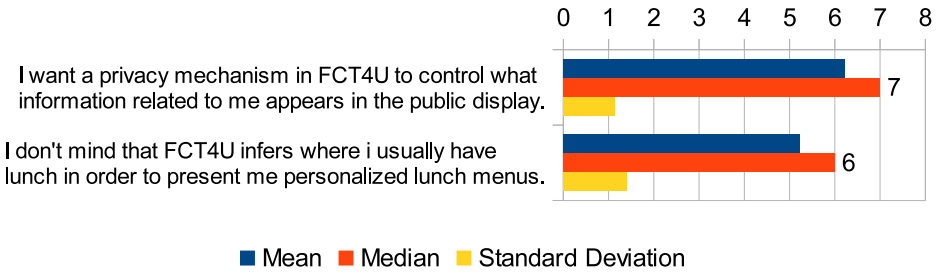


Figure 4.42: Privacy related questions

5

Conclusions and Future Work

This chapter presents conclusions of the current work, as well as solutions and ideas for future work that extends the contributions presented in this dissertation.

5.1 Conclusions

This work presents a Personalization Platform for Multimodal Ubiquitous Computing Applications (P^{2}MUCA), which can be used to assist third-party developers to personalize their applications according to each user and her/his current context. It was developed as a HTTP-based service, being secured by the OAuth 2.0 protocol. It can be applied to any class of application (e.g., mobile, desktop, web and other emerging types of applications) as long as an Internet connection is available. This approach saves third-party developers from having to develop specific solutions from scratch and allows them to outsource the development effort and computing power to a cloud-based solution.

The personalization model used by P^{2}MUCA comes from a previous work titled *Context-Aware Personalization Environment*, abbreviated as CAPE. Its model was integrated as if it was just another third-party library, but some modifications were made to add new features and fix bugs. This was not trivial because someone else's work had to be studied before integration and modifications could take place. Nevertheless, it was possible to reuse it almost entirely integrating into P^{2}MUCA.

The final solution consists of a service where developers can register their applications and end users authorize or revoke access to their data to specific third-party clients. This website still supports the XML-based configuration that was initially tested with CAPE, but also presents a newer user interface that allows users to more easily configure an application on the website. Additionally, it offers an API to application developers

that allows them to submit interaction stream data updates, user preferences updates, creation of new users and, of course, personalization requests.

The FCT4U system was developed as a multimodal ubiquitous application, being another major contribution of this work. It is multimodal because it supports multiple input and interaction mechanisms, i.e., passive presence, through mobile application, touch-screen and is prepared to include more options in the future. It is ubiquitous since the mobile application goes with users everywhere, while also being available through a situated public display at their work/study location.

FCT4U was developed as an application that could use P^{2}MUCA to help testing its development. The general goal was to provide useful information to faculty students and staff, while passing by a situated public display. Besides this initial objective, it ended up being a major source of interest to test and explore interaction with public displays, with a special highlight for the integration with mobile devices such as smartphones and tablets. In fact, a paper submitted to the UIC'13 conference during the development of this dissertation was mainly focused on the development of FCT4U.

By using P^{2}MUCA, FCT4U was able to adapt the content shown to each user, e.g., different pieces of information are ordered or omitted based on who is using the system, more sensitive information may also be hidden from view on the public display when the user is not alone or among friends, and a map may show traffic or public transport information depending of the user.

A fully functional prototype of FCT4U permitted the conduction of first lab study, which did not focused directly on personalization because that would need a much longer experience time. Instead, first and foremost was important to know if the system was going in the right direction by letting others experiment with it a give their personal opinion. Each user was briefed about the application and its overall goals, and was set to explore it by following three different interaction scenarios. Those scenarios focused on studying how users would react to the different interaction mechanisms provided, i.e., passive interaction through detection alone, interaction through the FCT4U mobile application and interaction by using the public display's touch-screen.

These tests also allowed to measure the interest on a system such as FCT4U and if the test subjects would be interested in having the information more tailored according to their preferences by letting the system learn while they were interacting with it. The responses were largely positive, providing some constructive criticism and suggestions. There was some apprehension regarding some more privacy sensitive features, such as the ones requiring to track a user's location, but in general they were enthusiastic about the idea of having information tailored to their habits and needs.

Therefore, in general, it was possible to verify that the work is going in the right direction by having a solid architecture based on a series of early design decisions, together with some user testing to help direct the project according to people's needs. With some of the ideas for improvements that are suggested in the next section, the system can surely move from a development testing environment to a fully public testing phase to later go

into production.

5.2 Future Work

Although all of the main objectives set out for this dissertation were met, there are still some areas that can be further improved and refined.

P^{2}MUCA mainly needs a better looking website, with a more modern, organized and appealing design. Given that the main focus was developing the platform and integrating CAPE into a readily available service, the web design part is something that was put on hold and that should be improved if the platform is to be launched to the general public. For example, the P^{2}MUCA Configurator was already developed with some extra care in terms of aesthetics by using Twitter Bootstrap that provides a basic aesthetically pleasant style. The rest of the website was built using the default templates provided by Spring Roo, which are not very visually appealing and look somewhat dated.

Besides a better web design, the application still needs to be stress tested with a (very) large number of simultaneous users to see how well it scales. There has been no big problems with a few dozens of users using the FCT4U application. Also, some synthetics tests that stressed CAPE directly were not a problem, even though it took a few seconds to process 1000 requests. Nevertheless, the ideal scenario would be to put FCT4U into a near-production environment, along with a few other applications at IMG, to test how P^{2}MUCA would cope under load.

Additionally, it would be interesting to change the deployment of the application on OpenShift from a non-scaled application to a scaled one, thus allowing to have multiple servlet instances serving the requests and removing the need of a SSH tunnel that connects the gears running P^{2}MUCA Website and P^{2}MUCA Service. If it is the database that proves to be the bottleneck, other database management systems (e.g., PostgreSQL) or database clustering and replication solutions can be used.

In order to further test and improve P^{2}MUCA as a whole, the underlying CAPE's personalization model may also need additional changes and tweaks. Something that should be improved is the expressiveness of the definition personalization options. The number of possible combinations grows exponentially when dealing with many parameters (e.g, FCT4U's *viewOrder* personalization in A.2), thus it would be useful to be able to define that a given parameter does not matter for a specific personalization option. This would cut the number of personalization options needed to cover the combinations of parameters that a developer may be interested in. Additionally, documentation should be added to the website so that third-party developers are able to understand and use the model to its fullest. Eventually, the platform may even become a commercial product if there is a demand for it.

Regarding FCT4U, it is important to make a few refinements based on the user feedback gathered during the first testing phase. In terms of widgets, the ones that need the most work are the Messages and News widgets. The Messages widget needs to be

rethought in order to be more appealing to users that already use other kinds of Internet-based communication. Maybe receiving notifications about new messages on the mobile device, which can then be seen when near the public display, would motivate users get close to it, or allowing messages to be sent to multiple users or groups of users, that could be manually defined or inferred due to usage patterns and connections between users (e.g., Facebook friends and groups). Such ideas could make the messaging widget more useful and appealing to users.

When it comes to the News widget, most complains come from the fact that it does not show very interesting information right now, because it only uses the FCT/UNL website RSS feed to show news titles. So, in the future, news should be mashed up from different sources, allowing users to choose from a list of interesting feeds or add their own. In fact, it may even be possible to use P^2MUCA to dynamically change the set of feeds shown to the user to match his profile, by using demographics and interaction data if no explicit preference is set.

Besides refining some of the widgets, users also expressed interest in having as much control as possible over their privacy, while they seem interested in features that require privacy sensitive information. For example, users were really interested in having the Lunch Menus widget showing first the place where they usually prefer to lunch, but they were also not very comfortable having the Android application tracking where they were all the time. Even though location tracking can be disabled completely, maybe users would not mind sharing their location only at some intervals. If users shared their location at lunch time (e.g., between 12h to 14h) than the system would still be able to guess their favorite lunch place. With that in mind, more fine grained preferences and privacy controls should be put in place, so that users are able to disable only the features they are not comfortable with.

Once these improvements are introduced, it is important to conduct a longer and broader user study to get results from a more representative sample of the potential users. Also, with a longer test, run the personalization model will have time to start to adapt itself to each different user, thus refining it according to user expectations will also be a major challenge during future testing phases. Considering that the touch-screen interface was the less favorite interaction mode during the first tests, and that it may be impractical in a more broad range, the next tests will focus on the passive interaction mode together with the use of the mobile application.

Additionally, multiple LCD screens were recently placed at various indoor points of the Department of Informatics' building to show useful information to users. One of them can eventually be used for the second testing phase since they are already placed in public areas of the building. In fact, considering that multiple displays are available, if the second testing phase is well received by the community, it may be interesting to adapt the application to run on all of the displays installed in the building, or maybe even expand it to other displays throughout the campus.

Bibliography

[Abe+10] F. Abel, I. I. Bittencourt, E. de Barros Costa, N. Henze, D. Krause, and J. Vassileva. "Recommendations in Online Discussion Forums for E-Learning Systems". In: *IEEE Transactions on Learning Technologies* 3.2 (2010), pp. 165–176.

[AMCed] P. Albuquerque, R. N. Madeira, and N. Correia. "FCT4U - When private mobile displays meet public situated displays to enhance the user experience". In: *UIC '13: Proceedings of The 10th IEEE International Conference on Ubiquitous Intelligence and Computing*. IEEE, 2013, Accepted.

[AGS11] H. Anderson, R. Gonda, and M. Strategy. *Facebook Open Graph and the Future of Personalization*. Tech. rep. SapientNitro, 2011. URL: `http://www.sapient.com/en-gb/sapientnitro/thinking/paper/26/facebook_open_graph_and_the_future_of_personalization.html`.

[And13] Andoid Developers. *Dashboard*. 2013 (Last Access: October 2013). URL: `http://developer.android.com/about/dashboards/index.html`.

[Ank+99] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. "OPTICS: Ordering Points to Identify the Clustering Structure". In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. SIGMOD '99. ACM, 1999, pp. 49–60.

[BP00] P. Bahl and V. N. Padmanabhan. "RADAR: An In-Building RF-Based User Location and Tracking System". In: *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 2. IEEE Computer Society, 2000, pp. 775–784.

[Bha09] S. Bhattacharya. "Place Identification: A Comparative Study Department of Computer Science". MA thesis. University of Helsinki, 2009.

[Blo10] J. D. Blower. "GIS in the Cloud: Implementing a Web Map Service on Google App Engine". In: *Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application*. COM.Geo '10. ACM, 2010, 34:1–34:4.

[Blu07] Bluetooth SIG. *Specification of the Bluetooth System (version 2.1 + EDR)*. Tech. rep. Bluetooth SIG, 2007. URL: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=241363.

[Boo+04] D. Booth, H. Haas, F. McCabe, E. Newcomer, I. M. Champion, C. Ferris, and D. Orchard. *Web Services Architecture*. Working Group Note. W3C, 2004. URL: http://www.w3.org/TR/ws-arch/.

[BR03] H. Brignull and Y. Rogers. "Enticing People to Interact with Large Public Displays in Public Spaces". In: *Proceedings of INTERACT' 03*. IOS Press, 2003, pp. 17–24.

[CMB08] X. Cao, M. Massimi, and R. Balakrishnan. "Flashlight Jigsaw: An Exploratory Study of an Ad-hoc Multi-player Game on Public Displays". In: *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work*. CSCW '08. ACM, 2008, pp. 77–86.

[Car06] J. Cardoso. "Enabling User Interaction in installation art using mobile devices". In: *British HCI Group Interfaces Magazine* 58 (2006), pp. 6–9.

[CJ12] J. Cardoso and R. José. "PuReWidgets: A Programming Toolkit for Interactive Public Display Applications". In: *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. EICS '12. ACM, 2012, pp. 51–60.

[Cha+08] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. "Bigtable: A Distributed Storage System for Structured Data". In: *ACM Trans. Comput. Syst.* 26.2 (2008), 4:1–4:26.

[Cha+06] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. "Bigtable: A distributed storage system for structured data". In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)*. 2006.

[CK02] Y. Chen and H. Kobayashi. "Signal strength based indoor geolocation". In: *Communications, 2002. ICC 2002. IEEE International Conference on*. Vol. 1. 2002, pp. 436–439.

[CSP09] J. Choi, B.-K. Seo, and J.-I. Park. "Robust Hand Detection for Augmented Reality Interface". In: *Proceedings of the 8th International Conference on Virtual Reality Continuum and Its Applications in Industry*. VRCAI '09. ACM, 2009, pp. 319–321.

[Cis09] Cisco Systems. *Cisco Cloud Computing - Data Center Strategy , Architecture , and Solutions - Point of View White Paper for U.S. Public Sector*. Tech. rep. Cisco Systems, Inc., 2009, pp. 1–16. URL: http://www.cisco.com/web/strategy/docs/gov/CiscoCloudComputing_WP.pdf.

[Cli13] S. Clinch. “Smartphones and Pervasive Public Displays”. In: *IEEE Pervasive Computing* 12.1 (2013), pp. 92–95.

[Cox+12] D. Cox, J. Wolford, C. Jensen, and D. Beardsley. “An Evaluation of Game Controllers and Tablets As Controllers for Interactive Tv Applications”. In: *Proceedings of the 14th ACM International Conference on Multimodal Interaction*. ICMI ’12. ACM, 2012, pp. 181–188.

[Cze+06] M. Czerwinski, G. Robertson, B. Meyers, G. Smith, D. Robbins, and D. Tan. “Large Display Research Overview”. In: *CHI ’06 Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’06. ACM, 2006, pp. 69–74.

[DH10] P. Dalsgaard and K. Halskov. “Designing Urban Media FaçAdes: Cases and Challenges”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’10. ACM, 2010, pp. 2277–2286.

[Dav13] J. Davies. *Great-Circle Distance*. 2013 (Last Access: August 2013). URL: `http://www.jasondavies.com/maps/distance`.

[Dav+09] N. Davies, A. Friday, P. Newman, S. Rutlidge, and O. Storz. “Using Bluetooth Device Names to Support Interaction in Smart Environments”. In: *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’09. ACM, 2009, pp. 151–164.

[Erb+08] A. Erbad, M. Blackstock, A. Friday, R. Lea, and J. Al-Muhtadi. “MAGIC Broker: A Middleware Toolkit for Interactive Public Displays”. In: *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*. PERCOM ’08. IEEE Computer Society, 2008, pp. 509–514.

[Erl05] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, 2005. Chap. 1, 3 – 5, 13 and 17. ISBN: 0131858580.

[Est+96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*. AAAI Press, 1996, pp. 226–231.

[Fac13] Facebook. *Facebook OpenGraph*. 2012 (Last Access: January 2013). URL: `https://developers.facebook.com/docs/opengraph`.

[Fet+11] I. Fette, I. "Google, A. Melnikov, and ". Ltd." *The WebSocket Protocol*. RFC 6455. "Internet Engineering Task Force (IETF)", 2011. URL: `http://tools.ietf.org/html/rfc6455`.

[Fie00] R. T. Fielding. “Architectural Styles and the Design of Network-based Software Architectures”. AAI9980887. PhD Thesis. University of California, Irvine, 2000.

[Fin+08] M. Finke, A. Tang, R. Leung, and M. Blackstock. "Lessons Learned: Game Design for Large Public Displays". In: *Proceedings of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts*. DIMEA '08. ACM, 2008, pp. 26–33.

[FPT12] R. Francese, I. Passero, and G. Tortora. "Wiimote and Kinect: Gestural User Interfaces Add a Natural Third Dimension to HCI". In: *Proceedings of the International Working Conference on Advanced Visual Interfaces*. AVI '12. ACM, 2012, pp. 116–123.

[FKK07] S. Frees, G. D. Kessler, and E. Kay. "PRISM Interaction for Enhancing Control in Immersive Virtual Environments". In: *ACM Trans. Comput.-Hum. Interact.* 14.1 (2007).

[GCG05] I. Garrigós, S. Casteleyn, and J. Gómez. "A Structured Approach to Personalize Websites Using the OO-H Personalization Framework". In: *Proceedings of the 7th Asia-Pacific Web Conference on Web Technologies Research and Development*. Vol. 3399. Lecture Notes in Computer Science. Springer-Verlag, 2005, pp. 695–706.

[Gar13] Gartner, Inc. *Gartner Says Smartphone Sales Accounted for 55 Percent of Overall Mobile Phone Sales in Third Quarter of 2013*. 2013 (Last Access: November 2013). URL: `http://www.gartner.com/newsroom/id/2623415`.

[Goo13a] Google Blog. *Search gets personal*. 2005 (Last Access: January 2013). URL: `http://googleblog.blogspot.pt/2005/06/search-gets-personal.html`.

[Goo13b] Google Blog. *Personalized Search for everyone*. 2009 (Last Access: January 2013). URL: `http://googleblog.blogspot.pt/2009/12/personalized-search-for-everyone.html`.

[Goo13c] Google Developers. *Google App Engine*. 2012 (Last Access: January 2013). URL: `https://developers.google.com/appengine`.

[Goo13d] Google Developers. *Google App Engine Premier Accounts*. 2012 (Last Access: January 2013). URL: `https://developers.google.com/appengine/docs/premier`.

[Goo13e] Google Developers. *Google App Engine Service Level Agreement*. 2012 (Last Access: January 2013). URL: `https://developers.google.com/appengine/sla`.

[Goo13f] Google Developers. *Google Cloud SQL*. 2013 (Last Access: January 2013). URL: `https://developers.google.com/cloud-sql`.

[Goo13g] Google Developers. *What is Google App Engine?* 2013 (Last Access: January 2013). URL: `https://developers.google.com/appengine/docs/whatisgoogleappengine`.

[Goo13h] Google, Inc. *Maps for mobile - My Location*. 2013 (Last Access: August 2013). URL: https://support.google.com/gmm/answer/1645548?hl=en.

[Goo13i] Google, Inc. *The Google Maps Geolocation API*. 2013 (Last Access: August 2013). URL: https://developers.google.com/maps/documentation/business/geolocation/.

[HB04] H. Haas and A. Brown. *Web Services Glossary*. Working Group Note. W3C, 2004. URL: http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/.

[Har+99] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. "The Anatomy of a Context-aware Application". In: *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*. MobiCom '99. ACM, 1999, pp. 59–68.

[Hic11] I. Hickson. *The WebSocket API*. Working Draft. W3C, 2011. URL: http://www.w3.org/TR/2011/WD-websockets-20110929/.

[HP10] J. Hollingsworth and D. J. Powell. "Teaching Web Programming Using the Google Cloud". In: *Proceedings of the 48th Annual Southeast Regional Conference*. ACM SE '10. ACM, 2010, 76:1–76:5.

[HKB08] E. M. Huang, A. Koster, and J. Borchers. "Overcoming Assumptions and Uncovering Practices: When Does the Public Really Look at Public Displays?" In: *Proceedings of the 6th International Conference on Pervasive Computing*. Vol. 5013. Pervasive '08. Springer-Verlag, 2008, pp. 228–243.

[HM03] E. M. Huang and E. D. Mynatt. "Semi-public Displays for Small, Co-located Groups". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Vol. 5. CHI '03. ACM, 2003, pp. 49–56.

[Hur+09] J. Hurwitz, R. Bloor, M. Kaufman, and F. Halper. *Cloud Computing For Dummies*. For Dummies, 2009. ISBN: 0470484705, 9780470484708.

[Int12a] Internet Engineering Task Force (IETF). *The OAuth 1.0 Protocol*. Tech. rep. IETF, 2012. URL: http://tools.ietf.org/html/rfc5849.

[Int12b] Internet Engineering Task Force (IETF). *The OAuth 2.0 Authorization Framework*. Tech. rep. IETF, 2012. URL: http://tools.ietf.org/html/rfc6749.

[Jan+05] M. Jansen, I. Uzun, H. U. Hoppe, and P. Rossmanith. "Integrating Heterogeneous Personal Devices with Public Display-Based Information Services". In: *Wireless and Mobile Technologies in Education, 2005. WMTE 2005. IEEE International Workshop on*. WMTE '05. IEEE Computer Society, 2005, pp. 149–153.

[Jos+08] R. José, N. Otero, S. Izadi, and R. H. R. Harper. "Instant Places: Using Bluetooth for Situated Interaction in Public Displays". In: *IEEE Pervasive Computing* 7.4 (2008), pp. 52–57.

[Kav+09] N. Kaviani, M. Finke, S. Fels, R. Lea, and H. Wang. "What Goes Where?: Designing Interactive Large Public Display Applications for Mobile Device Interaction". In: *Proceedings of the First International Conference on Internet Multimedia Computing and Service*. ICIMCS '09. ACM, 2009, pp. 129–138.

[Kha+05] A. Khan, J. Matejka, G. Fitzmaurice, and G. Kurtenbach. "Spotlight: Directing Users' Attention on Large Displays". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '05. ACM, 2005, pp. 791–798.

[Kic+97] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. "Aspect-Oriented Programming". In: *European Conference on Object-Oriented Programming, ECOOP 1997*. Vol. 1241. Invited presentation. Springer-Verlag, 1997, pp. 220–242.

[Kin13] Kinect Hacks. *Kinect Dressing Room*. 2011 (Last Access: January 2013). URL: `http://www.kinecthacks.com/kinect-dressing-room`.

[Koc05] M. Koch. "Supporting Community Awareness with Public Shared Displays". In: *Proc. Bled Intl. Conf. on Electronic Commerce*. 2005.

[KP07] A. Korth and T. Plumbaum. "A Framework for Ubiquitous User Modeling". In: *Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on Information Reuse and Integration*. IEEE Systems, Man, and Cybernetics Society, 2007, pp. 291–297.

[KO13] V. Kostakos and T. Ojala. "Public Displays Invade Urban Spaces". In: *IEEE Pervasive Computing* 12.1 (2013), pp. 8–13.

[Kub+13] T. Kubitza, S. Clinch, N. Davies, and M. Langheinrich. "Using Mobile Devices to Personalize Pervasive Displays". In: *SIGMOBILE Mob. Comput. Commun. Rev.* 16.4 (2013), pp. 26–27.

[Küp05] A. Küpper. *Location-Based Services: Fundamentals and Operation*. Wiley, 2005. Chap. 6 – 9. ISBN: 9780470092323.

[KHA12] E. Kurdyukova, S. Hammer, and E. André. "Personalization of Content on Public Displays Driven by the Recognition of Group Context". In: *Ambient Intelligence*. Vol. 7683. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 272–287.

[LY05] S. Lee and H.-S. Yong. "Web Personalization: My Own Web Based on Open Content Platform". In: *Proceedings of the 6th International Conference on Web Information Systems Engineering*. Vol. 3806. Lecture Notes in Computer Science. Springer-Verlag, 2005, pp. 731–739.

[Mad12] R. N. Madeira. "Personalization in pervasive spaces towards smart interactions design". In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*. IEEE Computer Society, 2012, pp. 548–549.

[Mad+14] R. N. Madeira, A. Vieira, P. Albuquerque, and N. Correia. "X-Personas: Building a Model for Personalization of Ubiquitous Computing Applications". In: *IUI '14: Proceedings of The 2014 International Conference on Intelligent User Interfaces*. ACM, 2014.

[Mal+04] E. Maler, J. Paoli, C. M. Sperberg-McQueen, F. Yergeau, and T. Bray. *Extensible Markup Language (XML) 1.0 (Third Edition)*. first Edition of a Recommendation. W3C, 2004. URL: http://www.w3.org/TR/2004/REC-xml-20040204.

[McD+08] D. W. McDonald, J. F. McCarthy, S. Soroczak, D. H. Nguyen, and A. M. Rashid. "Proactive Displays: Supporting Awareness in Fluid Social Environments". In: *ACM Trans. Comput.-Hum. Interact.* 14.4 (2008), 16:1–16:31.

[MG11] P. Mell and T. Grance. *The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standarts and Technology*. Tech. rep. 800-145. National Institute of Standards and Technology (NIST), 2011. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[MRS11] P. Merle, R. Rouvoy, and L. Seinturier. "A Reflective Platform for Highly Adaptive Multi-cloud Systems". In: *Adaptive and Reflective Middleware on Proceedings of the International Workshop*. ARM '11. ACM, 2011, pp. 14–21.

[Mue13] C. Muelle. *OPTICS: Ordering Points To Identify the Clustering Structure (Presentation)*. 2004 (Last Access: November 2013). URL: http://osl.iu.edu/~chemuell/projects/presentations/optics-v1.pdf.

[Nat13] National Coordination Office for Space-Based Positioning, Navigation and Timing. *GPS.gov*. 2013 (Last Access: August 2013). URL: http://www.gps.gov/.

[NNC95] National Research Council (U.S.). Committee on the Future of the Global Positioning System, National Academy of Public Administration, and Commission on Engineering and Technical Systems. *The Global Positioning System: A Shared National Asset*. National Academies Press, 1995, pp. 13–18. ISBN: 9780309052832.

[OAu13] OAuth. *About OAuth*. 2012 (Last Access: January 2013). URL: http://oauth.net/about.

[Ope13] OpenID Foundation. *Welcome to OpenID Connect*. 2013 (Last Access: September 2013). URL: http://openid.net/connect/.

[Ora13a] Oracle. *JavaFX Roadmap*. 2012 (Last Access: August 2013). URL: http://www.oracle.com/technetwork/java/javafx/overview/roadmap-1446331.html.

[Ora13b] Oracle Corporation. *MySQL Customers*. 2013 (Last Access: August 2013). URL: http://www.mysql.com/customers/.

[OW213] OW2 Consortium. *OW2 FraSCAti*. 2013 (Last Access: January 2013). URL: http://frascati.ow2.org.

[Oxf13] Oxford University Press. *Oxford Advanced Learner's Dictionary*. (Last Access: January 2013). URL: http://oald8.oxfordlearnersdictionaries.com/dictionary/personalize.

[Pae+04] T. Paek, M. Agrawala, S. Basu, S. Drucker, T. Kristjansson, R. Logan, K. Toyama, and A. Wilson. "Toward Universal Mobile Interaction for Shared Displays". In: *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*. CSCW '04. ACM, 2004, pp. 266–269.

[Pap08] M. P. Papazoglou. *Web Services: Principles and Technology*. Prentice Hall, 2008. Chap. 1 and 2. ISBN: 978-0-321-15555-9.

[PZL08] C. Pautasso, O. Zimmermann, and F. Leymann. "RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision". In: *Proceedings of the 17th International Conference on World Wide Web*. WWW '08. ACM, 2008, pp. 805–814.

[Pok+05] S. Pokraev, J. Koolwaaij, M. van Setten, T. Broens, P. D. Costa, M. Wibbels, P. Ebben, and P. Strating. "Service Platform for Rapid Development and Deployment of Context-Aware, Mobile Applications". In: *Proceedings of the IEEE International Conference on Web Services*. ICWS '05. IEEE Computer Society, 2005, pp. 639–646.

[Por13] Pordata. *Pordata: Base de Dados de Portugal Contemporâneo*. (Last Access: January 2013). URL: http://pordata.pt/.

[PCB00] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. "The Cricket Location-support System". In: *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*. MobiCom '00. ACM, 2000, pp. 32–43.

[Pri+01] N. B. Priyantha, A. K. Miu, H. Balakrishnan, and S. Teller. "The Cricket Compass for Context-aware Mobile Applications". In: *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*. MobiCom '01. ACM, 2001, pp. 1–14.

[Rap13] Raphael Leiteritz, Product Manager, Google. *Copy of Google's submission today to several national data protection authorities on vehicle-based collection of wifi data for use in Google location based services*. 2010 (Last Access: August 2013). URL: http://www.google.com/googleblogs/pdfs/google_submission_dpas_wifi_collection.pdf.

[Rim+11] B. P. Rimal, A. Jukan, D. Katsaros, and Y. Goeleven. "Architectural Requirements for Cloud Computing Systems: An Enterprise Cloud Approach". In: *Journal of Grid Computing* 9.1 (1 2011). 10.1007/s10723-010-9171-y, pp. 3–26.

[SAM13] SAML XML.org. *SAML Specifications*. Tech. rep. OASIS, 2013. URL: http://saml.xml.org/saml-specifications.

[Sch+08] T. Schlömer, B. Poppinga, N. Henze, and S. Boll. "Gesture Recognition with a Wii Controller". In: *Proceedings of the 2Nd International Conference on Tangible and Embedded Interaction*. TEI '08. ACM, 2008, pp. 11–14.

[SPD06] M. Sharifi, T. Payne, and E. David. "Public Display Advertising Based on Bluetooth Device Presence". In: *Mobile Interaction with the Real World (MIRW 2006) in conjunction with the 8th International Conference on Human Computer Interaction with Mobile Devices and Services*. 2006.

[spe07] specs@openid.net. *OpenID Authentication 2.0*. Tech. rep. OpenID Foundation, 2007. URL: http://openid.net/specs/openid-authentication-2_0.html.

[tut13] tutorialspoint. *What are Web Services*. (Last Access: January 2013). URL: http://www.tutorialspoint.com/webservices/what_are_web_services.htm.

[Twi13] Twitter, Inc. *MySQL at Twitter*. 2012 (Last Access: August 2013). URL: https://blog.twitter.com/2012/mysql-twitter.

[Vaq+08] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. "A Break in the Clouds: Towards a Cloud Definition". In: *SIGCOMM Comput. Commun. Rev.* 39.1 (2008), pp. 50–55.

[VC12] A. Vieira and N. Correia. "Context-Aware Personalization Environment for Mobile Computing". MA thesis. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2012.

[Vin75] T. Vincenty. "Direct and Inverse Solutions of Geodesics on the Ellipsoid with application of nested equations". In: *Survey Review* XXIII (misprinted as XXII) (1975), pp. 88–93.

[Wan+92] R. Want, A. Hopper, V. Falcão, and J. Gibbons. "The Active Badge Location System". In: *ACM Transactions on Information Systems* 10.1 (1992), pp. 91–102.

[Wei91] M. Weiser. "The Computer for the 21st Century". In: *Scientific American* 265.3 (1991), pp. 94–104.

[Wu+10] S. Wu, Y. Zhang, S. Zhang, X. Ye, Y. Cai, J. Zheng, S. Ghosh, W. Chen, and J. Zhang. "2D Motion Detection Bounded Hand 3D Trajectory Tracking and Gesture Recognition Under Complex Background". In: *Proceedings of the 9th ACM SIGGRAPH Conference on Virtual-Reality Continuum and Its Applications in Industry*. VRCAI '10. ACM, 2010, pp. 311–318.

[YJS06] A. Yilmaz, O. Javed, and M. Shah. "Object Tracking: A Survey". In: *ACM Comput. Surv.* 38.4 (2006).

[Yoo+12] H. Yoon, Y. Zheng, X. Xie, and W. Woo. "Social Itinerary Recommendation from User-generated Digital Trails". In: *Personal and Ubiquitous Computing* 16.5 (2012), pp. 469–484.

[YAS03] M. A. Youssef, A. K. Agrawala, and A. U. Shankar. "WLAN Location Determination via Clustering and Probability Distributions". In: *Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on*. PERCOM '03. IEEE Computer Society, 2003, pp. 143–150.

[Yua+08] M. Yuan, F. Farbiz, C. M. Manders, and K. Y. Tang. "Robust Hand Tracking Using a Simple Color Classification Technique". In: *Proceedings of The 7th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*. VRCAI '08. ACM, 2008, 6:1–6:5.

[ZZL12] W. Zeng, J. Zhao, and M. Liu. "Several public commercial clouds and open source cloud computing software". In: *Computer Science Education (ICCSE), 2012 7th International Conference on*. 2012, pp. 1130–1133.

[ZCB10] Q. Zhang, L. Cheng, and R. Boutaba. "Cloud computing: state-of-the-art and research challenges". In: *J. Internet Services and Applications* 1.1 (2010), pp. 7–18.

[Zho+04] C. Zhou, D. Frankowski, P. Ludford, S. Shekhar, and L. Terveen. "Discovering Personal Gazetteers: An Interactive Clustering Approach". In: GIS '04 (2004), pp. 266–273.

[Zho+07] C. Zhou, D. Frankowski, P. Ludford, S. Shekhar, and L. Terveen. "Discovering Personally Meaningful Places: An Interactive Clustering Approach". In: *ACM Trans. Inf. Syst.* 25.3 (2007).

[Zho+12] X. Zhou, Y. Xu, Y. Li, A. Jøsang, and C. Cox. "The State-of-the-art in Personalized Recommender Systems for Social Networking". In: *Artificial Intelligence Review* 37.2 (2012), pp. 119–132.

[ZSL05] A. Zimmermann, M. Specht, and A. Lorenz. "Personalization and Context Management". In: *User Modeling and User-Adapted Interaction* 15.3–4 (2005), pp. 275–302.

[Zyg11] E. Zygmuntowicz. *Cloud Foundry Blog - Cloud Foundry Open PaaS Deep Dive*. 2011. URL: `http://blog.cloudfoundry.com/2011/04/19/cloud-foundry-open-paas-deep-dive`.

Appendix A - Personalizations

Name	Parameters	Description
studentPublicTransport	userTypeProfile → *student*; transportProfile → *public*	A student that uses public transports
teacherPublicTransport	userTypeProfile → *teacher*; transportProfile → *public*	A teacher that uses public transports
studentPrivateTransport	userTypeProfile → *student*; transportProfile → *private*	A student that uses private transport
teacherPrivateTransport	userTypeProfile → *teacher*; transportProfile → *private*	A teacher that uses private transport
studentPublicTransportAlone	userTypeProfile → *student*; transportProfile → *public*; company → *alone*;	A student that uses public transports and is currently alone
teacherPublicTransportAlone	userTypeProfile → *teacher*; transportProfile → *public*; company → *alone*	A teacher that uses public transports and is currently alone

studentPrivateTransportAlone	userTypeProfile → *student*; transportProfile → *private*; company → *alone*	A student that uses private transport and is currently alone
teacherPrivateTransportAlone	userTypeProfile → *teacher*; transportProfile → *private*; company → *alone*	A teacher that uses private transport and is currently alone
studentPublicTransportAccompanied	userTypeProfile → *student*; transportProfile → *public*; company → *accompanied*;	A student that uses public transports and is currently accompanied
teacherPublicTransportAccompanied	userTypeProfile → *teacher*; transportProfile → *public*; company → *accompanied*	A teacher that uses public transports and is currently accompanied
studentPrivateTransportAccompanied	userTypeProfile → *student*; transportProfile → *private*; company → *accompanied*	A student that uses private transport and is currently accompanied
teacherPrivateTransportAccompanied	userTypeProfile → *teacher*; transportProfile → *private*; company → *accompanied*	A teacher that uses private transport and is currently accompanied
studentPublicTransportAccompanied w/friends	userTypeProfile → *student*; transportProfile → *public*; company → *accompanied w/friends*;	A student that uses public transports and is currently accompanied only with friends
teacherPublicTransportAccompanied w/friends	userTypeProfile → *teacher*; transportProfile → *public*; company → *accompanied w/friends*	A teacher that uses public transports and is currently accompanied only with friends

studentPrivateTransportAccompanied w/friends	userTypeProfile → *student*; transportProfile → *private*; company → *accompanied w/friends*	A student that uses private transport and is currently accompanied only with friends
teacherPrivateTransportAccompanied w/friends	userTypeProfile → *teacher*; transportProfile → *private*; company → *accompanied w/friends*	A teacher that uses private transport and is currently accompanied only with friends

Table A.1: *privacyMode* personalization options

Option Name & Parameters
weatherCommunicativeMenusNewsMapAholic • weatherAholic → *weatherAholic* • communicative → *communicative* • menusAholic → *menusAholic* • newsAholic → *newsAholic* • mapAholic → *mapAholic*
weatherCommunicativeMenusNewsMapAholicAlone • weatherAholic → *weatherAholic* • communicative → *communicative* • menusAholic → *menusAholic* • newsAholic → *newsAholic* • mapAholic → *mapAholic* • company → *alone*

weatherCommunicativeMenusNewsMapAholicAccompanied

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*
- company → *accompanied*

weatherCommunicativeMenusNewsMapAholicAccompanied w/friends

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*
- company → *accompanied w/friends*

weatherMenusNewsMapAholic

- weatherAholic → *weatherAholic*
- communicative → *notCommunicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*

communicativeMenusNewsMapAholic

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*

communicativeMenusNewsMapAholicAlone

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*
- company → *alone*

communicativeMenusNewsMapAholicAccompanied

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*
- company → *accompanied*

communicativeMenusNewsMapAholicAccompanied w/friends

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*
- company → *accompanied w/friends*

menusNewsMapAholic

- weatherAholic → *notWeatherAholic*
- communicative → *notCommunicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*

weatherCommunicativeNewsMapAholic

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*

weatherCommunicativeNewsMapAholicAlone

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*
- company → *alone*

weatherCommunicativeNewsMapAholicAccompanied

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*
- company → *accompanied*

weatherCommunicativeNewsMapAholicAccompanied w/friends

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*
- company → *accompanied w/friends*

weatherNewsMapAholic

- weatherAholic → *weatherAholic*
- communicative → *notCommunicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*

communicativeNewsMapAholicAlone

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*
- company → *alone*

communicativeNewsMapAholicAccompanied

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*
- company → *accompanied*

communicativeNewsMapAholicAccompanied w/friend

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*
- company → *accompanied w/friends*

newsMapAholic

- weatherAholic → *notWeatherAholic*
- communicative → *notCommunicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- mapAholic → *mapAholic*

weatherCommunicativeMenusMapAholic

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*

weatherCommunicativeMenusMapAholicAlone

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*
- company → *alone*

weatherCommunicativeMenusMapAholicAccompanied

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*
- company → *accompanied*

weatherCommunicativeMenusMapAholicAccompanied w/friends

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*
- company → *accompanied w/friends*

weatherMenusMapAholic

- weatherAholic → *weatherAholic*
- communicative → *notCommunicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*

communicativeMenusMapAholicAlone

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*
- company → *alone*

communicativeMenusMapAholicAccompanied

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*
- company → *accompanied*

communicativeMenusMapAholicAccompanied w/friends

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*
- company → *accompanied w/friends*

menusMapAholic

- weatherAholic → *notWeatherAholic*
- communicative → *notCommunicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*

weatherCommunicativeMapAholic

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*

weatherCommunicativeMapAholicAlone

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*
- company → *alone*

weatherCommunicativeMapAholicAccompanied

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*
- company → *accompanied*

weatherCommunicativeMapAholicAccompanied w/friends

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*
- company → *accompanied w/friends*

weatherMapAholic • weatherAholic → *weatherAholic* • communicative → *notCommunicative* • menusAholic → *notMenusAholic* • newsAholic → *notNewsAholic* • mapAholic → *mapAholic*
communicativeMapAholic • weatherAholic → *notWeatherAholic* • communicative → *communicative* • menusAholic → *notMenusAholic* • newsAholic → *notNewsAholic* • mapAholic → *mapAholic*
communicativeMapAholicAlone • weatherAholic → *notWeatherAholic* • communicative → *communicative* • menusAholic → *notMenusAholic* • newsAholic → *notNewsAholic* • mapAholic → *mapAholic* • company → *alone*

communicativeMapAholicAccompanied

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*
- company → *accompanied*

communicativeMapAholicAccompanied w/friends

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*
- company → *accompanied w/friends*

mapAholic

- weatherAholic → *notWeatherAholic*
- communicative → *notCommunicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- mapAholic → *mapAholic*

weatherCommunicativeMenusNewsAholic

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*

weatherCommunicativeMenusNewsAholicAlone

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*
- company → *alone*

weatherCommunicativeMenusNewsAholicAccompanied

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*
- company → *accompanied*

weatherCommunicativeMenusNewsAholicAccompanied w/friends

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*
- company → *accompanied w/friends*

weatherMenusNewsAholic

- weatherAholic → *weatherAholic*
- communicative → *notCommunicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*

communicativeMenusNewsAholic

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*

communicativeMenusNewsAholicAlone

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*
- company → *alone*

communicativeMenusNewsAholicAccompanied

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*
- company → *accompanied*

communicativeMenusNewsAholicAccompanied w/friends

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*
- company → *accompanied w/friends*

menusNewsAholic

- weatherAholic → *notWeatherAholic*
- communicative → *notCommunicative*
- menusAholic → *menusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*

weatherCommunicativeNewsAholic

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*

weatherCommunicativeNewsAholicAlone

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*
- company → *alone*

weatherCommunicativeNewsAholicAccompanied

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*
- company → *accompanied*

weatherCommunicativeNewsAholicAccompanied w/friends

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*
- company → *accompanied w/friends*

weatherNewsAholic

- weatherAholic → *weatherAholic*
- communicative → *notCommunicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*

communicativeNewsAholicAlone

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*
- company → *alone*

communicativeNewsAholicAccompanied

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*
- company → *accompanied*

communicativeNewsAholicAccompanied w/friend

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*
- company → *accompanied w/friends*

newsAholic

- weatherAholic → *notWeatherAholic*
- communicative → *notCommunicative*
- menusAholic → *notMenusAholic*
- newsAholic → *newsAholic*
- Aholic → *notAholic*

weatherCommunicativeMenusAholic

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*

weatherCommunicativeMenusAholicAlone

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*
- company → *alone*

weatherCommunicativeMenusAholicAccompanied

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*
- company → *accompanied*

weatherCommunicativeMenusAholicAccompanied w/friends

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*
- company → *accompanied w/friends*

weatherMenusAholic

- weatherAholic → *weatherAholic*
- communicative → *notCommunicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*

communicativeMenusAholicAlone

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*
- company → *alone*

communicativeMenusAholicAccompanied

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*
- company → *accompanied*

communicativeMenusAholicAccompanied w/friends

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*
- company → *accompanied w/friends*

menusAholic

- weatherAholic → *notWeatherAholic*
- communicative → *notCommunicative*
- menusAholic → *menusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*

weatherCommunicativeAholic

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*

weatherCommunicativeAholicAlone

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*
- company → *alone*

weatherCommunicativeAholicAccompanied

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*
- company → *accompanied*

weatherCommunicativeAholicAccompanied w/friends

- weatherAholic → *weatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*
- company → *accompanied w/friends*

weatherAholic

- weatherAholic → *weatherAholic*
- communicative → *notCommunicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*

communicativeAholic

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*

communicativeAholicAlone

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*
- company → *alone*

communicativeAholicAccompanied

- weatherAholic → *notWeatherAholic*
- communicative → *communicative*
- menusAholic → *notMenusAholic*
- newsAholic → *notNewsAholic*
- Aholic → *notAholic*
- company → *accompanied*

communicativeAholicAccompanied w/friends • weatherAholic → *notWeatherAholic* • communicative → *communicative* • menusAholic → *notMenusAholic* • newsAholic → *notNewsAholic* • Aholic → *notAholic* • company → *accompanied w/friends*
none • weatherAholic → *notWeatherAholic* • communicative → *notCommunicative* • menusAholic → *notMenusAholic* • newsAholic → *notNewsAholic* • Aholic → *notAholic*

Table A.2: *viewOrder* personalization options