Masterarbeit:
Adaptionsunterstützung für die CRUISe
Laufzeitumgebung

by
Ruíz Díaz Carlos Alberto

A thesis submitted to the
Institut für Systemarchitektur,
Lehrstuhl Rechnernetze
in partial fulfilment of the
requirements for the degree of
Master of Science

Fakultät Informatik
Technische Universität Dresden

March 2010

Dresden, Germany
Abstract

In recent years, the effects of Moore’s law are apparent everywhere: Chip density, processor speed, memory cost, disk capacity, and network bandwidth are improving more and more\cite{1}. All these changes are not only applied to personal computers, like laptops or desktop computers but also to small mobile devices such as Palmtops and smart phones. All these lead people to be able to access their personal information, and work data, whenever they want and wherever they are, the so called ”anytime, anywhere” approach. Nowadays, people wants to be more independent and keep on moving, without sacrificing the advantages that modern life provides them. Therefore applications that can supply services according to the needs and wishes of the user are more suitable for mobile environments. Because of these reasons, we present an approach that focuses on the definition and development of User Interface Services, which should be able to adapt themselves according to the user preferences and device mobile capabilities. The information about the user and mobile device is considered as the context information our application should be aware of. It is also important that such services have to perform adaptations whenever a change on the context information happens, i.e. at runtime.
Acknowledgements

To my friends, family and specially to my dear three Mothers.
# Contents

Abstract ii  
Acknowledgements iii  
List of Tables viii  
List of Figures ix  

1 Introduction 1  
  1.1 Motivation ................................................. 2  
  1.2 Objectives .................................................. 3  
    1.2.1 CRUISe project ......................................... 4  
  1.3 Use case ................................................... 4  
    1.3.1 Mobile Travel Guide Application Scenario ............. 5  
    1.3.2 Adaptation Scenario ................................... 5  
  1.4 Thesis Structure .......................................... 6  

2 Related Work 7  
  2.1 Portable Help Desk (PHD) ................................... 8  
  2.2 Multiple Concern Adaptation for Run-time Composition in Context-Aware Systems ................................................... 9  
  2.3 Middleware Demonstrator Server (MIDAS) ................. 9
2.4 Context-Aware Middleware for Pervasive and Ubiquitous Service (CAM-PUS). .................................................. 10
2.5 Preference Model for Context Aware Applications. ............... 12
2.6 Context-Aware Middleware and Human Interaction Markup Language (HIML). .................................................. 13
2.7 Context-Aware and Pervasive Networks (Capnet). ................. 16

3 Foundations .......................................................... 19
3.1 Context Definition ................................................. 19
   3.1.1 Hardware Model ........................................... 20
   3.1.2 User Model .................................................. 21
3.2 Mobile Collaboration Architecture - ”MoCA” ..................... 21
   3.2.1 MoCA Functionality ....................................... 23
   3.2.2 Mobile Collaboration Architecture Web Service .......... 25
3.3 CroCo: Ontology-Based, Cross-Application Context Management ... 28
   3.3.1 CroCo Architecture. ...................................... 29
   3.3.2 CroCo Functionality. ...................................... 31
3.4 Ext JS Javascript Framework and RIA Platform. .................. 33
3.5 Mobile Travel Guide Platform .................................... 35

4 General Architecture for Context Aware Adaptations ............. 38
4.1 Overview ......................................................... 40
4.2 Client Side ...................................................... 41
   4.2.1 Service Holder. ........................................... 42
   4.2.2 Services. .................................................. 44
   4.2.3 CRUISe Monitor. ......................................... 45
4.3 Server side ...................................................... 45
   4.3.1 Context Watcher and Context Service. .................... 46
      4.3.1.1 Service Expression Registry. ....................... 47
4.3.1.2 Device Authentication. ................................. 47
4.3.1.3 Device Authentication Data Base. ...................... 48
4.3.2 Web Service Information Retriever (WSIR). ................. 48
4.3.2.1 Data Converter. .................................. 48
4.3.3 Context Information Service (CIS) Server ................... 49
4.3.4 Data service provider. ................................ 49
4.3.4.1 General Information Web Services. ..................... 49
4.3.4.2 Context Information Web Service. ..................... 49
4.4 Communication model. .................................... 50
4.4.1 Inter-client side communication. ......................... 51
4.4.2 Client - Server Communication. .......................... 51
4.4.3 Server - Data service provider communication. .......... 52
4.5 Adaptations. ............................................. 53
4.6 Context Model. .......................................... 55

5 Implementation. ............................. 57
5.1 Services and Adaptations. ................................ 57
5.2 User profiles. ........................................... 64
5.3 Server side and Context Providers. .......................... 65
5.3.1 Context Watcher and Context Service. .................... 65
5.4 Web Service Information Retriever. ......................... 66

6 Evaluation ............................... 68
6.1 Test environment. ...................................... 68
6.1.1 Server. ............................................. 68
6.1.2 Monitors. ........................................... 70
6.2 Test Descriptions. ...................................... 71
6.2.1 General Use Case. .................................... 71
6.2.2 Test 01: Sport profile user. ............................ 72
6.2.3 Test 02: Business profile user ............................ 72
6.2.4 Test 03: Tourist profile user ............................. 73
6.3 Results ......................................................... 73

7 Conclusion ...................................................... 77
List of Tables

4.1 Self adaptive UI services ........................................... 39
4.2 Context Model ......................................................... 56
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>CAMPUS Architecture.</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Context- and preference-based decision process for context-aware applications.</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>Context-Aware Middleware System Architecture.</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>Component networks. Capnet Architecture.</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>MoCA Architecture.</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>MoCA Interactions.</td>
<td>24</td>
</tr>
<tr>
<td>3.3</td>
<td>MoCA/WS Architecture.</td>
<td>25</td>
</tr>
<tr>
<td>3.4</td>
<td>Communication between clients, MoCA/WS and MoCA Architecture.</td>
<td>26</td>
</tr>
<tr>
<td>3.5</td>
<td>SyncTimeOut Approach.</td>
<td>27</td>
</tr>
<tr>
<td>3.6</td>
<td>Async Approach.</td>
<td>28</td>
</tr>
<tr>
<td>3.7</td>
<td>CroCo Architecture.</td>
<td>30</td>
</tr>
<tr>
<td>3.8</td>
<td>CroCoOn Ontology.</td>
<td>31</td>
</tr>
<tr>
<td>3.9</td>
<td>CroCo workflow.</td>
<td>32</td>
</tr>
<tr>
<td>3.10</td>
<td>Ext js Architecture.</td>
<td>34</td>
</tr>
<tr>
<td>3.11</td>
<td>Ext Toolbar example.</td>
<td>34</td>
</tr>
<tr>
<td>3.12</td>
<td>Input of required parameters and results</td>
<td>36</td>
</tr>
<tr>
<td>3.13</td>
<td>Mobile Travel Guide Architecture.</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>Architecture Overview.</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>Client Side Architecture.</td>
<td>42</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In recent years, the effects of Moore’s law are apparent everywhere: Chip density, processor speed, memory cost, disk capacity, and network bandwidth are improving more and more[1]. All these changes are not only applied to personal computers, like laptops or desktop computers but also to small mobile devices such as Palmtops and smart phones. All these lead people to be able to access their personal information, and work data, whenever they want and wherever they are, the so called ”anytime, anywhere” approach.

Distributed systems that assume a stationary execution are not longer suitable for such mobile scenarios. Many mobile computing researchers deal with these kind of scenarios by adapting existing distributed systems[4]. But this approach is not really suitable, because the modified systems cannot really be adapted to the limited computational resources of small mobile devices.

Another idea is to provide mobile-aware applications. Such applications can be more effective and adaptive without consuming too much attention of the user, but only if they can take advantage of dynamic environmental characteristics, such as user’s location, time of the day, light levels as well as the hardware context data like battery and connectivity levels.

Context-aware applications should be able to react to their environment and mod-
ify themselves, either on their layout or their internal operation, in order to provide
a better usage experience to the application user.

Nowadays, people want to be more independent and keep on moving, without
sacrificing the advantages that modern life provides them. Therefore applications that
can supply services according to the needs and wishes of the user are more suitable
for mobile environments.

1.1 Motivation

Mobile applications are a rapidly developing segment of the global mobile market[2].
They consist of software that runs on a mobile device and performs certain tasks
for the user of the mobile device, using information about the device and its capa-
bilities. Mobile applications are distinctly different from browsing the Mobile Web
which is still characterized by latencies due to page load times (albeit there are some
trends around JavaScript/Ajax and mobile widgets which will cross over between
both worlds, i.e. providing fast response times once loaded). However, mobile appli-
cations should be designed according to the specific capabilities of the mobile device
and after installation on the mobile device is difficult to modify them according to the
user needs. Hence, we consider developing a Mobile Web application which takes into
consideration the device capabilities (device context) and user needs (user context)
to provide services that suit to the user preferences and not depend directly on a
specific mobile device, but dynamically adapts to the mobile device. Therefore, the
main idea of this work is:

"Web applications should adapt to the user needs, not the other way around".
1.2 Objectives

The main goal of this Thesis is to provide User Interface Services (UIS) as an improvement to the Mobile Travel guide platform[6], such services should be able to adapt themselves according to the current user context at runtime, taking into consideration the user preferences or requirements as well as the characteristics of the used mobile device.

The adaptations should be performed following predefined adaptations rules and mechanisms, which have been defined for each service.

In order to gather information about the context of the mobile device the ”Mobile Collaboration Architecture”[14] middleware is used, which provides access to inner mobile device data. Also some other web services are used, which provide relevant information to the user, such as weather or news information.

With the union of all these elements the final result should be able to extend the Mobile Travel guide platform with self adaptive services that user really wants or needs.

Because of these reasons, our approach has to focus on the definition and development of User Interface Services, which should be able to adapt themselves according to the user preferences and device mobile capabilities. The information about the user and mobile device is considered as the context information our application should be aware of. It is also important that such services have to perform adaptations whenever a change on the context information happens, i.e. at runtime. In order to control which context information is relevant to our approach a context model should be designed. It has to consider different user and device attributes as context variables, for example the location of the user, his language and the energy levels of the mobile device. It is also important to design an architecture which supports these services and provides them with the required context information whenever it is available, as well as with service specific information, e.g. information about the weather and restaurants.
1.2.1 CRUISe project

Parallel to the development of the Internet, the devices which access the internet have changed dramatically. Web developers are faced with an unprecedented breadth of devices, from cell phones to multi-monitor desktop computers. Due to the large number of new applications and user groups and their increasing mobility, Web applications are now very different and changeable.

In order to meet the increased demands and requirements of Web-based user interfaces will require new approaches for their development, deployment and maintenance. The central challenge is the dynamic, context-dependent selection, configuration and integration or composition of complex, application-independent UI components. The result is a series of scientific challenges:

- Establish a basic component model for encapsulating and re-use of Web-based UI components.
- Development of models and languages to describe Web-based UI components.
- A description of the composition of user interfaces, particularly regarding the control of the data and control flow.
- Dynamic, context-dependent selection, configuration, invocation and integration of UI components

1.3 Use case

Before to give a description about the adaptation scenario of this work, it is necessary to give a short introduction to the Mobile Travel Guide Application scenario, because our approach extends the functionality of the Mobile Travel Guide Platform.
1.3.1 Mobile Travel Guide Application Scenario

The Mobile Travel guide platform, provides to the user travel services, like automatic route suggestions (from a starting point to a destination) to Points-of-Interest (POI) location according to the destination.

The application scenario described by the travel guide platform is:

Users have to set a start point and a destination, with this information the mobile travel guide platform have to provide route suggestions as well as some other information about the destination, such as:

- Hotels.
- Events of interest to the user.
- Different transportation options.

A map of the desired place will be used to display all the information; a text based form will be also used to display more detailed information about Hotels and transportation options.

1.3.2 Adaptation Scenario

The adaptation scenario considers different kinds of users, such as Tourists, Business men or people interested in sports; therefore users need really useful information according to their preferences and location. Also we have to consider the device characteristics.

Some important characteristics about the device to take into consideration to perform an adaptation are:

- Limited computing power.
- Limited energy supply.
• Limited internet connection.

Because of these limitations, a service should be able to modify itself to display only the most important information or even to stop its operations if required. Some adaptation examples would be:

• Display only text with the information about the route and not a graphical way if the signal strength is weak.

• Stop streaming service if battery is low.

This is the adaptation scenario proposed for this work.

1.4 Thesis Structure

In subsequent chapters we present the approach we have developed. Thus, this work is structured as follows: In chapter 2, we present a survey of related works given a short description about them and their main characteristics. Chapter 3 focuses on the different supporting systems (context providers, javaScript frameworks), that are used in order to develop our approach. Chapter 4 presents the design and architecture of our approach and the integration with the supporting systems. Chapter 5 and 6 deal with details about the implementation and evaluation of our approach. Finally, chapter 7 presents the conclusions about our work.
Chapter 2

Related Work

In this chapter, we survey previous research approaches about context awareness, giving a short description of them, their context infrastructure and, in some cases, how context adaptations are performed.

In [4] is presented a Survey about mobile context aware systems and applications, which incorporate context information to themselves for performing several adaptations or modifications to their own structure. The classification of context aware applications this survey presents is based on how a system performs adaptations based on current contextual information. The following classification is presented:

- **Proximate selection**, a user-interface technique where the objects near to the user current location are emphasized in a two dimensions screen, making them easier to interpret.

- **Automatic contextual reconfiguration**, a process of adding new components, removing existing components, or changing the connections between components due to context changes.

- **Contextual information and commands**, a technique where same commands (user instructions) produce different results depending on the context in which they are executed.
• **Context-triggered actions**, IF-THEN actions rules used to specify how context-aware systems should adapt.

This classification, presents different perspectives on how a mobile application can take advantage of context.

Hereafter are presented approaches which propose either an application, framework or architecture model.

### 2.1 Portable Help Desk (PHD).

In this section we describe an ad-hoc application designed by the Carnegie Mellon University, which basically provides location information about class schedules, coffee shops, restaurants and other places or events for the students within the university campus. Their approach is called Portable Help Desk (PHD)[1] and it was designed to provide two fundamental services: **spatial awareness** and **temporal awareness**.

The spatial awareness service is a service which uses a wireless access point network to set up, within the campus, all the possible locations for students. When a student access the PHD through one of this access points, his current location is sent to a central server which has a registry of all access points locations, informing the students their current locations. The temporal awareness service provides information to students based on the day time and their schedule. Therefore, as a requirement, before using the PHD system, the students have to register their activities into the PHD database. This project was an approach to provide information taking into consideration some context variables. However, it has some drawbacks such as: (i) The information is only accessible to pre configured access points, i.e. if an access point is not registered at the central server, it is not possible to get either location information or scheduled information; and (ii) The application does not perform adaptations according to the context data, it just shows information, which has been already registered in the central server.
2.2 Multiple Concern Adaptation for Run-time Composition in Context-Aware Systems.

This approach is focussed on the modification of system components which can be affected by changes in context conditions, such as: user location, time and resource availability at run-time. The main purpose of this approach is to present an idea to perform composition and adaptation of COTS (Commercial-Off-The-Shelf) components at run-time[3]. In this approach, such components are used to construct larger systems saving time, effort and money. However it is necessary to perform some minor adaptations in the components to integrate them with the rest of the system. They claim that such adaptations can be more efficient than adaptations that only consider generic integration issues, if they consider the current context where the system is executing. Hence, they provide an approach to perform such adaptations at runtime instead of compile-time, modifying the components interfaces. However, this approach only focuses on component composition of a system and adaptations which can be applied to these components, and not in the adaptation of user interfaces.

2.3 Middleware Demonstrator Server (MIDAS).

The generic framework described in this work, provides support for the development of a customized middleware, reconfigurable stacks and adaptive application services[11]. The framework allows the reconfiguration of a system, based on context information provided by sensors or actuators (devices which converts measurements of physical quantities into signals which can be read by an observer or by an instrument). This is possible because all components provided by the framework are generic, i.e. they do not depend on any specific hardware, but they can be modified by the developer according to its own needs. This gives to the application developed with the framework enough flexibility, because the developer does not have to write specific
components from scratch, but only reuse the provided components by the framework and if necessary modify the component or develop an own one. To allow calibration, i.e. the modification of a context component at run-time, the technical framework specifies a context element as a reconfigurable service. They are implemented as a container for a self-describing data format, like for example a XML document. With this calibration support, even the adaptation logic can be reconfigured. As the framework supports generic adaptation, it is possible to rearrange abstract function roles (sensors, actuators and interpreters) that can hide any kind of technical realization. This work gives us an idea of how the generic components can be implemented to allow adaptations based on context information.

2.4 Context-Aware Middleware for Pervasive and Ubiquitous Service (CAMPUS).

This work presents an ontology-based model that is used to facilitate the middleware layer with the ability to understand and reason the semantics of entities deployed on it, and to thereby enable the middleware-driven run-time adaptations decisions. With the implementation of middleware-driven run-time adaptation decisions, they do not force developers from the prediction, formulation and maintenance of adaptation rules; likewise they give the possibility to achieve an optimized quality of service by deferring the adaptation decisions to run-time. Figure 2.1 presents an image of CAMPUS architecture[17].

A set of ontologies, called CAMPO, were designed. CAMPO makes possible to understand the underlying semantics of the information provided by various contexts. It is split in two dimensions: Domain and Range. The first dimension is divided into:

- Context ontologies. Model various context entities to share contextual information in various domains.
Component ontologies. Capture concepts and roles in the domain of the development and deployment of components. They expose the computational and coordinative semantics of components, and help the middleware select the most appropriate components to assemble target applications.

Application ontologies. They are used to describe the properties of target context-aware applications.

And the second dimension into:

Foundation ontologies. These are models of common objects in the middleware, which are generally applicable across a wide range of domain-specific ontologies.

Domain specific ontologies. Model a particular domain that is a part of the physical world. They represent the particular meanings of terms as they apply to that domain.
This allows to CAMPUS to be more extensible and scalable, and also helps to improve the system performance.

### 2.5 Preference Model for Context Aware Applications.

The approach presents a Preference model, which supports user-customisable decision-making[7], figure 2.2 presents how the decision process is performed. In the decision-making process the user preferences are evaluated against a set of context information, candidate choices and application state variables, yielding an assignment of ratings to the candidate choices. The user preferences may reflect the requirements of one or multiple users and an arbitrary kind of choices can be supported.

![Figure 2.2: Context- and preference-based decision process for context-aware applications.](image)

To support this Preference model a layered software infrastructure was develop. This infrastructure provides support for:

- Integration, management and querying of context information from various sources, including sensors, context-aware applications and human users (Context management layer).
• Management and evaluation of user preference information (preference management layer).

• Decision making and branching at the application layer, using the services of the context and preference management layers.

Also some guidelines to develop user interfaces are presented, such as:

• Constrain types of personalisation that can be performed by users.

• Integrate personalisation mechanisms into the everyday use of the application.

• Provide logging and feedback mechanisms.

• Provide useful default behaviours.

The presented approach is able to use context information from different sources modifying its internal behaviour based on user preferences. However, it is not possible the modification of the user interface layout, only the user preferences parameters. User interfaces will remain inalterable even if the context information changes.

2.6 Context-Aware Middleware and Human Interaction Markup Language (HIML).

Here middleware architecture is presented focusing on the user interaction using a new XML based interface language called HIML (Human Interaction Mark-up Language)[8]. HIML was designed to describe and construct context messages among context aware agents and store them into a database for later use. The design objectives of the middleware can be explained as:

• Each context aware agent in each device contributes to constitute context information and it is also able to access the stored context information.
• User can continuously monitor current context information and control it manually.

Figure 2.3: Context-Aware Middleware System Architecture.

The core of this architecture approach is the Context Manager, which manages and controls context information of a specific service environment such as home, hospital and conference. The middleware works as follows: each device agent reports its context information to the system, after gathering the Context information, this is structured into a consistent form following the HIML standard defined by the architecture. After all context information is restructured in HIML form, it is stored
in a context database. The accumulated context knowledge is fused and analyzed through data mining; this information is used to form new contexts. Once new context are formed, they are stored into the context database and used to activate the operation of terminal devices. The generated context can be used to generate new context as well. Figure 2.3 shows the described architecture.

The presentation manager of the architecture, allows the user to set customized configuration values at any time and at any place obtaining efficient and detailed control of the system, by adjusting some parameters manually through the user interface provided by the presentation manager. This feedback given by the user will generate new messages which will be transformed into HIML and transferred to the context manager to produce new contexts. The interaction with the user is managed through the presentation manager component. With the generation of new context the application can be adapted to the user necessities; however this is possible only after a considerable number of user contexts have been stored into the context database.

**Context Model.**

The approach considers time as the main context variable. The HIML architecture describes events based on time, it defines conditions where the time is specified, once this time is reached and action is performed. To define actions information about the target of the action (mobile device and person) as well as the action itself (what has to be performed) have to be defined. They consider information about the person and the mobile device as part of the context but not as context variables, because they do not monitor the current state of the mobile device or the person to perform an action.
2.7 Context-Aware and Pervasive Networks (Capnet).

This approach presents an architecture which offers a set of generic services for building context-aware mobile applications to mobile users. The architecture, Context-Aware and Pervasive Networks (Capnet)[13], divides functionality into components of different domains. Components can be searched, downloaded, and started as required at run-time. This allows the modification of the application structure according to the current situation. In this approach such components are location transparent and hence they communication transparently. Capnet architecture also specifies a service to obtain context information, either in asynchronous or synchronous way, retrieving the context information as context events. Some other features this architecture provides are: data storage that associates documents with context, dynamic and transparent connectivity adaptation, context-sensitive service discovery and asynchronous messaging between applications. Figure 2.4 shows some examples of how Capnet Architecture divides functionality into components.

The main goal of this approach was to provide a wide set of services collected into a single architecture, the wide usage of context, and the ability of services to adapt to the changes in the environment. Hereafter, a brief description of the architecture components is given:

- **Component and Component Management.** Component is the smallest distinguishable and identifiable part of the architecture. A component description specifies all resources a component needs; such specification is written in a XML format. Component management allows the communication between components.

- **Service Discovery.** It provides functionality for search and finding other components and services. It uses different service discovery protocols, depending
on their availability and suitability for the current situation.

- **Connectivity Management.** Connectivity adaptation is realized by maintaining communication channels. The channel approach helps to hide complexity of network infrastructure and allows dynamic adaptation of the network services; therefore the connectivity management component monitors the context, including the status of the network and adapts the channel’s connections.

- **User Interface.** It provides support for the design and implementation of application UIs using three different techniques: abstract UIs (XML), plug-in UIs (downloadable code) and web-based UIs (html). This was achieved by separating the application from its UI implementation using an XML-based UI script language to describe the abstract UI elements and their properties. The UI component is responsible for managing the user preferences and modifying the UI to conform to context changes that affect the look and feel of the UI. A whole UI architecture was designed to this purpose.
• Media. It provides a unified interface for handling media over different platforms, which enables developers to focus on the media instead of the various underlying platforms.

The approach described is an architecture which integrates all context concerns into a single application, the UI part is important from the point of view of our project because it focuses on run-time adaptations of user interfaces of the services they provide to the user.
Chapter 3

Foundations

3.1 Context Definition

The word context is defined as "the interrelated conditions in which something exits or occurs" in Merriam-Webster’s Collegiate Dictionary\(^1\). This definition tends to be vague in a computing environment, however it has been used in many ways in different areas of computer science.

In [4] is stated that context in mobile computing has two different aspects:

- One includes the characteristics of the surrounding environment that determine the behavior of the mobile applications (active context).

- The second aspect is relevant to the application but not critical because it does not influence the behavior of the application directly, therefore is not necessary for applications to adapt to this kind of context (passive context).

Taking these two aspects into consideration they define context as:

> “Context is the set of environmental states and settings that either determines an application’s behavior or in which an application event occurs and is interesting to the user.” [4]

\(^1\)http://www.m-w.com
It is also said that there are two ways of how an application can take advantage from context, therefore they define two kinds of context aware applications:

- “Active context awareness: an application automatically adapts to discovered context, by changing the application’s behavior.” [4]
- “Passive context awareness: an application presents the new or updated context to an interested user or makes the context persistent for the user to retrieve later.” [4]

For our approach we use the previously given context definition, but we want to emphasize the difference between hardware related-context (Hardware model), that related exclusively with the mobile device, and user related-context (User model), environmental events related exclusively with the user and his preferences. Also according to the given classification of context-aware applications, our approach is classified as an ”active context-aware application”.

### 3.1.1 Hardware Model

Our approach considers all events generated by a mobile device as part of the hardware model, events such as:

- The quality of wireless connection.
- Remaining energy.
- CPU usage.
- Free memory.
- Current Access Point (AP).

All these kind of event can not be controlled by the user of the mobile device, even if the user is aware of the occurrence of these events, he can not modify the configuration of the mobile device in order to change the way in which these events occurs. Therefore, an application that considers the hardware related-context should be able to adapt itself to these changes.
3.1.2 User Model

The User model concerns with information about the user, i.e. events about information the user is interesting in. Therefore, the user model should consider characteristics about a person, characteristics such as:

- Age.
- Person impairments.
- Language.
- Geographic information.

These kind of characteristics can be controlled and modified by the user in order to fulfill his personal preferences. This leads to events that are user specific and should to be considered by context aware applications in order to improve its behavior.

3.2 Mobile Collaboration Architecture - ”MoCA”

The Mobile Collaboration Architecture (MoCA) [14] was designed for infrastructured wireless networks. MoCA provides client and server APIs, basic services supporting collaborative applications and a framework for implementing application proxies, which can be customized to specific needs of the application in order to facilitate the access to the basic services. Figure 3.1 depicts the MoCA Architecture.

The APIs and basic services have been designed to provide support for different types of applications, e.g. applications with synchronous or asynchronous interaction, message-oriented applications.

In MoCA each application has three parts: a server, a proxy and a client, where the first two execute on a static node in the wired network, while the client runs on a mobile device. One or more proxies of the application are the intermediates of any communication between the server and the client components. The proxy of an
application may execute several tasks, such as adaptation of transferred data, e.g. compression, context processing, distribution of context information. The proxy also serves as a mean of distributing the application-specific processing among the server and its proxies.

As depicted in figure 3.1 the main services offered by the architecture are the following:

**Monitor.** Is a daemon executing on each mobile device, which is responsible for collecting state information of the mobile device, such as connectivity quality, free memory, CPU usage, and sending this data to the Context Information Service (CIS), executing on the wired network.

**Configuration Service (CS).** This service is in charge of storing and managing the configuration information for all mobile devices, so that these can use the MoCA infrastructure. The configuration information is stored in a persistent database, where
each entry holds the following data: MAC address of the device, the address (IP and Port) of the CIS and Discovery Service, and the periodicity in which the monitor will send information about the state of the device to the CIS.

**Context Information Service (CIS).** This service receives and processes state information sent by the monitor. It also receives request for notifications (subscriptions) from application proxies, and generates and delivers events to a proxy whenever a change in the state of the device is of interest to the proxy.

**Discovery Service (DS).** Is in charge of storing, managing and locating information regarding any application registered with the MoCA middleware.

**Location Information Service (LIS).** This service infers the approximate symbolic location of a device (SRM), using a specific context information collected by CIS: the pattern of RF signal strength received from all nearby Access Points.

**Symbolic Region Manager (SRM).** LIS allows the administrator to define logical regions of arbitrary size and rectangular shape and a hierarchical description of regions and their nested sub-regions. It is the SRM who implements this functionality and also provides an interface to define (create, modify, delete) and request information about the symbolic regions.

**APIs and the ProxyFramework.** The server and the client of an application should be implemented using the MoCA APIs. The proxy of the application will be an instance of the ProxyFramework. Both the APIs and the ProxyFramework hide from the application developer all details concerning the use of the services provided by MoCA.

### 3.2.1 MoCA Functionality

Here is described the sequence of interactions between the elements of the MoCA architecture.

Initially, the application server registers itself at the DS informing the name and the properties of the service that it implements, each proxy contained in the applica-
Figure 3.2: MoCA Interactions.

The application should perform the same action to inform the DS about their location. Once the application server and proxies are registered, each mobile device starts executing the MoCA monitor, which sends information about the device to the CIS. Information about the address of CIS and the periodicity in which information about the device is sent, is provided to the monitor by the CS when the device is started. When all these information is set, the monitor starts sending state information to the CIS. The clients of the server application start making requests once they have discovered a Proxy which implements a service they are interested in. These requests are routed through the corresponding Proxy, which processes and forwards them to the application server. Every proxy is in charge of send request to the CIS for registering its interest in notification of events concerning the client it represents. To register an interest to the CIS, the proxy uses an SQL-like sentence which specifies the context variable and the condition that such context variable has to satisfy. Whenever the CIS receives information about the current state of the device, it checks whether this state evaluates any expression to true, if so, the CIS sends a notification to all proxies.
which have registered such expression.

The processing of the context information depends on the application requirements. All the context information can be used to determine the actions the application server or client should perform when a threshold or condition is met[15]. Figure 3.2 shows these interactions between MoCA components.

The MoCA architecture implements mobility transparency for the applications. This is achieved with the help of the Proxy, which performs a handover at the application level, whenever the mobile device moves to a new network.

### 3.2.2 Mobile Collaboration Architecture Web Service

MoCA Web service was designed in order to allow non-Java client programs to access and use the context services provided by the MoCA architecture[10].

![MoCA/WS Architecture](imageURL)

Figure 3.3: MoCA/WS Architecture.

This web service acts as a proxy and offers a similar interface to its clients to
that offer by MoCA architecture APIs. MoCA/WS offers methods to access only the Context Information Service (CIS) and Location Information Service (LIS), because these are consider as the main context services offered by the MoCA architecture. Figure 3.3 depicts the overall architecture of MoCA/WS.

**Functionality.** When a client application sends a request to the MoCA/WS (through a SOAP message over HTTP), it forwards this request to the MoCA service and returns a SOAP reply.

MoCA/WS and MoCA establish a CLIENT-SERVER relationship, so MoCA/WS is responsible for handling client requests, while the MoCA is only responsible for handling one client the MoCA/WS. Synchronous requests to the MoCA/WS works exactly equal as any other Java application that uses the MoCA APIs, however because of its nature MoCA/WS is stateless therefore, asynchronous requests do not work as with normal Java clients that use MoCA APIs. Figure 3.4 shows these interactions between the client. MoCA/WS and MoCA architecture.

![Figure 3.4: Communication between clients, MoCA/WS and MoCA Architecture.](image)

To deal with asynchronous requests the MoCA/WS implements: **SyncTimeOut** and **Async methods.**

The **SyncTimeOut** offers access to the asynchronous methods with a condition. A client does a request to the MoCA/WS passing as argument a timeout, this timeout is checked and the response is sent back when an event occurs or the timeout occurs. Hence, there is a chance that the client receives a null value as answer. If the client wants to use LIS services, there is another argument, which is the number of events
desired, if the number of events is reached before the timeout happens, the events are returned to the client. If the timeout happens before the number of events is reached, the MoCA/WS returns all the events that were received (even if it is not the desired number of events). Figure 3.5 shows this approach.

![SyncTimeOut Approach](image)

The other method is called *Async*. It uses the concept of tickets, where a ticket holds the response of an asynchronous request. On request the MoCA/WS creates a ticket that associates with the events received by the MoCA architecture, this ticket is sent back to the client. All the events associated with a ticket are stored in a virtual table within the MoCA/WS. The client can request events at anytime with the corresponding ticket and decides whether MoCA/WS should keep storing events or not. Figure 3.6 shows how this approach works.
3.3 CroCo: Ontology-Based, Cross-Application Context Management

The Cross application Context manager Service (CroCo) is an ontology-based generic system [12] that allows context providers to submit and context consumer to request context data via specific service interfaces. With the help of consistency and inference rules, CroCo can derive high-level information that may not be provided by any sensor and also CroCo serves as a general context supplier for external context-aware applications.

The CroCo system has as main requirements, which are the requirements a Cross-application context management must fulfill, the followings:

- It must allow diverse applications, applications plug-ins, software agents and sensors to act as context providers or consumers.
- It needs to offer a generic and flexible mechanism for dynamic registration of
such agents.

• Registered consumers should be able to fetch (query) relevant data from the service at any time (pull), or be notified if relevant data managed by the service has changed (push).

• Context consistency needs to be ensured, therefore, special care needs to be taken regarding security and privacy issues.

The CroCo system follows the Blackboard model [9], which promotes a data-centric approach in which external processes can submit context information, or subscribe for change notifications. With this is possible to add new context providers and consumers.

3.3.1 CroCo Architecture.

CroCo consists of three main parts: data management, consistency checking and reasoning and context update and provision. Figure 3.7 depicts the CroCo Architecture.

Data Management. It has three main components: the Context History (CH), which contains the history of updates to the context model, the Consistent Context (CC), representing the currently valid, consistent contextual data, and the Inferred Knowledge (IK) this one encapsulates all information reasoned from the current context information. The Context History and the Inferred Knowledge form the Context Store, all clients request information to the Context Store so its internal division into CC and IK is transparent to the clients. The CH is used internally for statistical analysis or by context reasoners, hence it is completely hidden.

The Consistency Manager performs consistency checking every time new contextual data is added to the context model. Any number of Consistency Enforcers can register at the CM to carry out consistency checks and conflict detection.
Reasoning is performed by a **Reasoning Manager**, as well as with CM an arbitrary number of Reasoner can register at the RM, which invokes them to start the reasoning process once relevant data changes. Context Reasoning is based on facts stored in a **Facts Database** and its results are subject to subsequent consistency checks. All inferred data is stored separately in the IK base.

To support context data update and provisioning, CroCo provides two services: **Update Service and Query Service**. The Update Service facilitates updates and changes to the context model, while the Query Service provides two ways to retrieve context information from CroCo, Synchronous and Asynchronous way.

**CroCo Context Model.**

A generic, ontology-based context model called **CroCoOn (Cross-application Context Ontology)**(fig.3.8), was developed for the use within CroCo. It consist of several sub-ontologies that model different aspects of context. Based on this upper ontology it is possible to extend the model and integrate domain-specific knowledge.
to facilitate the usage of CroCo. This extensions are called *Ontology Profiles* as they describe a certain type of activity more in detail. Thus, they represent the common conceptualization of context providers and consumers for a particular application scenario.

3.3.2 CroCo Functionality.

Hereafter is described how the components of CroCo work together at run time during context updates and requests. These interactions are depicted in Figure 3.9.

1. First a context provider sends new data in form of regular RDF triples to the Context Update Service.

2. The context information is forwarded to the Context Store, which internally manages the consistent and inferred context. It uses CroCoOn to model persons (users), groups, time, etc.

3. To ensure that the submitted context information is consistent with the internal
model, it is validated by the Consistency Manager.

4. A consistency report is returned to the Context Store, which is added to the Context History together with the submitted context data.

5. If the data is validated as consistent, the reasoning process is started by the Reasoning Manager.

6. The knowledge inferred by the reasoning process is again checked for consistency and stored in the Context History.

7. The Callback handler is informed that the context model has changed.

8. Previously registered Context consumers are notified of the context data changes.

9. This notification about changes is only performed if the Privacy Enforcer grants the authorization.
3.4 Ext JS Javascript Framework and RIA Platform.

Ext JS is a cross-browser client-side JavaScript framework [5] for building rich internet applications. It was created by Jack Slocum in 2006, as an extension for the Yahoo! User Interface (YUI) library. The version 1.0 of the library was officially released on April 1, 2007.

It supports all major web browsers, such as:

- Internet Explorer 6+.
- Firefox 1.5+ (Pc and Mac).
- Safari 3+.
- Opera 9+ (Pc and Mac).

As a client-side framework, Ext can run against any server platform that can process POST requests and return structured data, such as: PHP, Java and .NET. Former versions of Ext required one of the following base libraries to be included: YUI\(^2\), jQuery\(^3\) or Prototype/Script.aculo.us\(^4\). Ext contains adapters that provide some of the basic plumbing utilities from those libraries, including Ajax support, animation, DOM manipulation, event handling. Beginning with version 1.1, Ext includes a native Ext adapter, so the external libraries are no longer required(Figure 3.10). The Ext adapter provides some support to integrate external libraries; however, it is no guaranteed that all functions of Ext work properly. Only YUI, jQuery and Prototype/Script.aculo.us are fully supported because a specific Ext adapter is provided for each library.

Ext Js includes:

\(^2\)http://developer.yahoo.com/yui/
\(^3\)http://jquery.com/
\(^4\)http://www.prototypejs.org/
Customizable UI widgets, such as: Windows, Trees, Charts, Layout Managers, Forms. (Figure 3.11).

Extensible component model.


Figure 3.10: Ext js Architecture.

Figure 3.11: Ext Toolbar example.
3.5 Mobile Travel Guide Platform

Mobile Travel Guide [6] is a mobile web application which main purpose is to provide to the user a route from a starting point to a destination. It also offers some other functionalities such as: find points of interest, hotels and events near to the destination selected by the user. A route can be found depending on the type of transportation selected by the user, there are three main types of transportation: Bus, Airplane and Train.

Use Case.

The Mobile Travel Guide defines a use case where a person wants to find a route to a destination as well as an Hotel near to the desired destination. To achieve this, the user has to specify an starting point and a destination. Once the user has specified these two points, he should select which kind of transportation he desires, there are three types available: Bus, Train or Airplane, it is possible to select one or all of them. The user can also select what kind of places he is interested in. He can select social events that are near to the destination, as well as hotels and conferences.

Once these parameters are defined, the Mobile Travel Guide application searches for possibles routes based on the starting point, destination and the selected type of transportation, together with the routes, the desired places and events near to the destination are searched. If a route is found, it is displayed on a map. Additional information about the route is displayed in a text form. In the same way, information about points of interest near to the destination are showed, they are presented in the map where the destination is specified and not where the route is displayed.

Architecture.

The Mobile Travel Guide application consists of three parts (Figure 3.13):

- A Javascript-based front-end.

- A Java back-end server.

- And external service which provide the required information.
The javascript front-end is in charge of parameter setting, i.e. the user selects the information he wants through the graphic interface provided by the front-end. Figure 3.12 shows how the required information is set.

![Figure 3.12: Input of required parameters and results](image)

The Java back-end server consists of two parts: **The Request Handler and the Data Retriever.**

The **Request Handler** is in charge of handling requests and responses from and to the client. Therefore, it provides a generic interface to the client, this interface supports the different kinds of request that come from the user. The communication between the client and the Request Handler is performed through asynchronous RPC calls, these because the client side should not be blocked while a request to the server is performed.

The **Data Retriever** is in charge of forwarding all the request from clients to the corresponding external services. Hence, it should transform the client request into an appropriate request to the external service, or if needed, scrap the web page where the required information is.

The external services provides the information the user requires, these are services
such as:

- Google maps. Provides information about geographical places.
- Last.fm service. Provides information about Conferences, Hotels and Social Events.
- ÖPNV. It is a company which provides information about the transportation system of a city (schedules, lines and connections). This service is used to compute a route from one place to another.

Figure 3.13: Mobile Travel Guide Architecture.
Chapter 4

General Architecture for Context Aware Adaptations

As stated before, our approach is classified as an active context aware application (an application that automatically adapts to the context information), therefore all adaptations on its services should be done at runtime when an event of interest to the user or service occurs. The application not only uses the already mentioned context providers\(^1\), but also some other web services, which provide different information to the user, such as:

- Google Maps. It provides faculties for map manipulation.
- Google Calendar. It provides facilities for calendar manipulation, such as: appointment creation, coordinating meetings and events, or scheduling work shifts.

With the combination of these web services, the application will provide to the user the services described in the Service Table 4.1.

This chapter describes the design what was followed to implement such services. First an overview of the entire architecture is presented. Subsequent sections focus

\(^1\)See chapter number 3. Foundations
Table 4.1: Self adaptive UI services

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;The closest place&quot;</td>
<td>Finds the closest geographical places of interest for the user.</td>
</tr>
<tr>
<td>&quot;Weather forecast&quot;</td>
<td>Finds the weather forecast information to specific places.</td>
</tr>
<tr>
<td>&quot;Personalized news&quot;</td>
<td>Receives the News of a specific country.</td>
</tr>
<tr>
<td>&quot;Automatic route calculation for Agenda Appointments&quot;</td>
<td>Calculate the route to the address of the next appointment in the agenda.</td>
</tr>
<tr>
<td>&quot;Automatic ticket buying&quot;</td>
<td>Finds sale points based on the proximity of the device to special buying places.</td>
</tr>
<tr>
<td>&quot;Cheap Site seeing&quot;</td>
<td>Finds routes which cover a city with public transportation.</td>
</tr>
<tr>
<td>&quot;I am in a hurry&quot;</td>
<td>Calculates the fastest route to a location.</td>
</tr>
<tr>
<td>&quot;Special eating&quot;</td>
<td>Finds special kinds of Restaurants.</td>
</tr>
</tbody>
</table>
on detailed descriptions of each component of the architecture, such as: client and server sides, as well as the communication model we use.

4.1 Overview

Our approach consists of three main parts: an end user part, the server part, and the data service provider part.

The architecture has such structure because it provides a modular approach, thus more flexible. We design different modules which provide a specific functionality to our architecture, for example a module which communicates with the different context providers and modules which communicate with the client. Because of the design, we make available general interfaces for each component which are independent of their implementation. This allows us to modify the implementation of certain functionality without modifying others components of the architecture.

In general, the Client Side of the architecture contains services which provide information to the user and also perform adaptations according to the current context in behalf of the user. The Client Side also provides several profiles depending on user preferences and it is in charge of the device information acquisition.

The Server side is in charge of handling client requests, either requests about context information or requests to external services such as Wetter.com (weather information) and Google Calendar (appointment information). It handles device information, which is acquired from the client side, and performs the authentication of the device identifiers. The Server side is also in charge of the context tracking, i.e. observes when the context information requested by the user changes using a publish - subscribe model.

The third part of the architecture, Data Service Provider part, consists of web services we use in order to provide information for a service. Web services such as: MoCA web service, Google Web service and Wetter.com web service provide
context information and general information to a client service. Figure 4.1 presents an overview of our architecture.

![Architecture Overview](image)

Figure 4.1: Architecture Overview.

A Context Model which considers user and hardware attributes is also part of our approach. It is designed in order to accomplish our approach requirements and it is based on the context models provided by MoCA and CroCo architectures.

### 4.2 Client Side

The Client Side of the architecture consists of three main parts:

- Service Holder.
- Service.
- CRUISe Monitor.

As depicted in Figure 4.2. Hereafter a detailed explanation of each component and its functionality is given.
4.2.1 Service Holder.

The Service Holder contains all services required by the user. This component is in charge of handling the communication between the services and the module in charge of handling context information requests in the server side (Context Watcher\(^2\)), i.e. it gets all the requests (context information a service requires) from services and forwards them to the corresponding module in the server side, in the same way it gets the context information from that server side module and forwards it to the service that requires this information. Information that a service requires, which is not part of the context information, is handled by the service itself, i.e. the service asks for information to the module in charge of non-context information requests (Web Service Information Retriever\(^3\)).

In order to perform this Service - Context Watcher communication, the Service

\(^2\)Context Watcher is explained in this chapter on Server side section.
\(^3\)Web Service Information Retriever is explained in this chapter on Server side section.
Holder should be able to maintain a registry of context information needed by the services. Therefore, when a service is created it is registered at the Service Holder and after that the Service Holder requests the expressions (an expression consists of a context variable and a condition) the service wants to register. All of the expressions are transformed into requests to the Context data provider on the server side, that each expression belongs to a request. Requests to the Context data provider contain the following data: expressions, service identifiers and a mobile device identifier.

When context data from the Context Watcher is received by the service holder, this searches into its registry which service registered an expression for that data and forwards it to the specified service, once the service holder has forwarded the data to the service, the expression is removed from the service holder registry. With this if the same data is received twice by the service holder the second time the data is not forwarded to the service, so that the same adaptation is not performed twice.

We can summarize the behavior of the Service Holder as follows: obtain the device identifier specified by the user and the expressions that each service wants to register. With the information about expressions and services, the Service Holder creates a temporary register which contains Service Expression value pairs. These value pairs are used to construct requests to the Context Watcher, the request consists of a Service Expression value and the device identifier. After the Service Holder constructs the requests, it sends them to the Context Watcher and waits for the response of each one of the expressions. When a response arrives, the Service Holder gets the service or services contained in the response and searches them in its registry, when the Service Holder finds all services, it sends the context data to them, i.e. the expression which has been met. Finally the Service Holder deletes the already accomplished expressions from its registry.
4.2.2 Services.

They implement certain functionality. They are also responsible for implementing adaptations. As depicted in the figure 4.2, a Service contains, apart from service specific elements, Expressions and Adaptation Rules.

- An Expression represents the context information required by the Service. It consists of a context variable and a condition for such variable. An expression can be composed by one or many expressions, when many expressions are presented it is necessary to indicate a logical operator between the expressions, the logical operators can be either AND or OR. With the specification of these operators it is possible to decide whether all component expression are necessary to fulfill the compose expression or not.

- An Adaptation Rule is responsible for the execution of adaptations on the service whenever an Expression is met. Adaptation Rules can be one of two types:
  - Functional Adaptation Rule which modifies the way in which the service works.
  - Layout Adaptation Rule which modifies the way the service is displayed.

A Service can contain zero or many Expressions, and each of them can have one or many Adaptation Rules (Fig. 4.3); Expressions without Adaptation Rules are not allowed because is pointless requesting context information if this is not utilized. The cardinality between Expressions and Adaptation Rules can be one of the followings:

- Expression (1: 1) Adaptation rule.
- Expression (1 : N) Adaptation rule.
- Expression (N : 1) Adaptation rule.
4.2.3 CRUISe Monitor.

This component continuously sends information about the current state of the mobile device to the CIS Server (placed in the Server Side). Its main functionality is to gather context information about the client mobile device and user context. It runs as a daemon on the mobile device. The collected data includes the quality of the wireless connection, remaining energy, CPU usage, free memory, current Access Point (AP), list of all APs and their signal strengths that are within the range of the mobile device.

4.3 Server side

As depicted in figure 4.4 the Server Side consists of several components, the main components of the server side in our approach are:
4.3.1 Context Watcher and Context Service.

The Context watcher is in charge of forwarding requests from client side to the Context service and vice versa, i.e. The purpose of the Context Watcher is to track context expression made by the service, in order to determine whether or not an expression has been satisfied. It also keeps a registry about the Expression it has received and the Services the Expression belongs to (similar to Service holder in client side).
The Context Service provides a common and transparent interface to access the services offered by MoCA and CroCo context providers. When it receives a request from the Context Watcher, the Context Service redirect this request either to the MoCA architecture or to the CroCo architecture. This architecture selection is transparent to the Context Watcher, because this is internally done by the Context Service.

When context data from the Context Service is received by the Context Watcher, this searches which expression has been met in its registry and generates a response to the Service Holder, the response contains the Expression which has been met plus the Service(s) that asked for that expression. In the case of hardware information, the response does not contain a specific value about the current state of the mobile device, i.e. no an explicit value is received about the current state of the mobile device, only information about which expression has been satisfied is received, in this way neither the service holder nor the service needs to validate the received value in order to see if an adaptation needs to be performed.

Before the response to Service Holder has been sent, the Expression is removed from the Context Watcher registry.

4.3.1.1 Service Expression Registry.

The Service Expression Registry is used to keep a registry of the Expressions sent by the Service Holder and the Services that subscribed such Expressions. Each entry within the Service Expression Registry is store as Expression - Service value pair. Each entry can be associated to more than one service, this because many services can subscribe the same expression with the Context Watcher. Hence, only one request to the Context Service is performed even if the Expression has more than one Service.

4.3.1.2 Device Authentication.

The Context Watcher is also responsible for the authentication of the mobile device, before it starts any interaction with the Context Service, the Context Watcher has
to check whether the device identifier sent by the service holder is already in use, if the case, the Context Watcher does not initiate any interaction between the Context Watcher and the Context Service, otherwise the interaction occurs as explained before. In both cases is possible to use the services, however if the mobile device is already in use, the Services are not able to perform any adaptation.

4.3.1.3 Device Authentication Data Base.

This Data Base stores the device identifier and the session in which the device identifier is registered. The data base keeps information about the device only while the user session is active, whenever the user finishes the interaction with the Context Watcher (either because the application was closed by the user or because of a time-out), the device identifier and the session values stored in the database are erased.

4.3.2 Web Service Information Retriever (WSIR).

It is in charge of forwarding Services request to the corresponding web service, i.e. it retrieves all the information needed by the Service to work, information such as: weather forecast, restaurants and touristic places. After a request is received, the WSIR transforms the information from the Service into an appropriate request to the desired web service. When any web service responds the request, the WSIR converts the response from the web service into a readable Service response, i.e. into data that a service can interpret, after the data conversion ends, the WSIR forwards the data to the Service.

4.3.2.1 Data Converter.

This component is used by the Web Service Information Retriever in order to transform a web service response into a Service Readable data. The Data Converter takes
the information from a web service response converts this information into a standard data representation which all Services can understand and sends the transformed data to the WSIR. In this way the WSIR does not have to apply conversions to the response data format of web services.

### 4.3.3 Context Information Service (CIS) Server

The Context Information Service stores all device context information, which comes from the CRUISe Monitor on the client side\(^4\). The CIS server is able to response to client requests either in synchronous or asynchronous way, so the client can decide if fetch for information or wait until a event of his interest happens.

### 4.3.4 Data service provider.

This section is divided into General Information Web Services and Context Information Web Service (MoCA Web Service).

#### 4.3.4.1 General Information Web Services.

The general information web services provide different information depending on the needs of the service, for example information about weather, restaurants, hotels and places of interest to the user. All these web service are offered by distinct Companies and no information other than how to access them is provided.

#### 4.3.4.2 Context Information Web Service.

The Mobile Collaboration Architecture Web Service (MoCA/WS) provides context information about the hardware of a specific device\(^5\). Information about its functionality and the way how it is implemented were already described.

---

The Cross-application Context Management (CroCo) is used to handle information about the user context. Information about its functionality and the way how it is implemented were already described.

4.4 Communication model.

This subsection explains how the communication between the different parts of the architecture is performed and also the communication protocols, because the context data communication protocol differs from the general information communication protocol. In general, communication between the client side and the server side is done through RPC calls (the RPC protocol allows the execution of a specified procedure in a remote server using supplied parameter, the RPC protocol is used in order to maintain the standard defined by the Mobile Travel Guide platform), and the communication between the server side and the data service providers is done using SOAP protocol and HTTP protocol, this because some of the data service providers uses HTTP as communication protocol. Figure 4.5 presents the Communication Model of the Architecture. Most of data service providers rely on SOAP protocol as mean for information exchange; because of this the server should perform this kind of communication with them. Each data provider has a counterpart on the server side (these are enclosed by the Web service information retriever and Context service) which is in charge of the communication between the data provider and the server.

---

7Simple Object Access Protocol is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks
8Hypertext Transfer Protocol is an application-level protocol for distributed, collaborative, hypermedia information systems.
4.4.1 Inter-client side communication.

To perform the communication between the different Services an event based communication is used. When a Service is created, this subscribes to the Service Holder the events it wants to make available to other Service. A Service can also add listeners to specific events it wants to receive. When a Service produces an event this is forwarded to the Service Holder and this forwards the event to all other Services, the event is sent from the Service Holder to Service using a broadcast approach and only the Service that have a listener for this specific event reacts to the event.

4.4.2 Client - Server Communication.

As stated before there are two entities which are in charge of the communication between the client and the server: the Service Holder and the Context Watcher.

The Service Holder (client side) and the Context Watcher (server side) handle context data communication, this is performed using a "Long-polling" approach. The Long-polling is a technique that optimizes traditional polling to reduce latency. It sends a request to the server, but a response is not returned to the client until one is
available. As soon as the connection is closed, either due to a response being received by the client or a request time out, a new connection is initiated[16]. Long-polling communication is performed until there are no more expressions in the Service Holder registry or in the Context Watcher registry.

The communication between Services and Web Service Information Retriever is performed using asynchronous HTTP requests (AJAX\textsuperscript{9} requests).

### 4.4.3 Server - Data service provider communication.

Communication between general information web services and server is performed using SOAP messages or HTTP requests, depending on the protocol they support. As stated before, the Web Service Information Retriever is responsible for performing such communication.

Communication between the Context Service and MoCA web service is through SOAP messages, MoCA web service offers a Publish-Subscribe communication model (asynchronous communication), and that is the model used by our server to request context information about a specific mobile device. The Context Service generates one subscription to MoCA web service for each expression that has been received. Publish-Subscribe is an asynchronous messaging paradigm where senders (publishers) of messages are not supposed to send their messages to specific receivers (subscribers). Subscribers express interest in one or more topics, and only receive messages that are of interest, without knowledge of what publishers there are. As well as with MoCA web service, the communication between CroCo and the Context Service is through SOAP messages in an asynchronous way.

\textsuperscript{9}Asynchronous JavaScript and XML
4.5 Adaptations.

This section describes how an Adaptation is performed by our approach. First we give an example of an adaptation and taking the example as reference, we explain the whole adaptation process.

We consider the service which calculates the shortest route from a starting point to a destination, depending on the strength of the connection signal, the service can either display the information about the route in a map or display it only as text. The key points of the example are: required context information (signal strength) and the adaptation to perform (display information on a map or as text). Figure 4.6 shows the structure of this example on the client side.

![Figure 4.6: Entities on client side for shortest route example.](image)

Therefore, after a Service registers an Expression (Context Information) with
the Service Holder, the Service waits for an Expression event to be satisfied (the event generated by Service Holder when a Context condition is met). Because is not guaranteed that will be ever satisfied, while waiting for an Expression event the Service functionality is not blocked. When the Service receives an Expression event it executes the corresponding Adaptation Rule.

If there is more than one adaptation rule to be performed by an Expression, the Service firstly executes all Functional Adaptation Rules and secondly Layout Adaptation Rules. We consider that the execution of functional adaptation rules has priority over layout adaptation rules because a functional adaptation rule can lead to the modification of the service layout. This process is repeated until only one Adaptation Rule remains. All adaptation rules are specified by the services, when an adaptation rule is described within the service, it is necessary to specify which context information the adaptation rule needs in order to be executed, also which components of the service are going to be modified and how each component is going to be modified. An adaptation rule, either Functional or Layout, follows these steps in order to perform a service modification.

- First the Adaptation Rule verifies if all necessary components to perform the adaptation are available. These components are specified when an adaptation rule is defined.

- If all components are available, the Adaptation Rule performs the adaptation specified for such component; otherwise the Adaptation Rule ends without performing any modification.

This kind of verification allows us to perform several adaptations on the same, without interfering with other adaptation rules which perform adaptations on the same components. Therefore, Adaptation Rules are considered as independent entities. The assumption is that if an Adaptation Rule does not find all the previously specified components, it is because this Adaptation Rule was already executed.
In order to perform an adaptation two main steps are consider: Make a Subscription for Context Information and Handling Context Information.

Subscription. In this step a service subscribes an expression with the service Holder (the context information the service needs). After all expressions are subscribed, the service holder sends this information to the server, which in is in charge of subscribing this expression with the context providers and also tracking this expression, in order to verify when the expression is satisfied.

Context Information Handling. Once an expression is satisfied, the server sends information about the expression to the service holder, this send the data to the corresponding services, which verify what adaptation rules, should be performed and which service components should be adapted, following the already explained sequence.

4.6 Context Model.

Our approach considers the context model as a composition of two parts: the hardware model, which considers information related to the mobile device, and the user model which takes into consideration information related to the user. It is based on the context models provided by MoCA and CroCo architectures.

The context model created for the context of this work provides the following context attributes:
Table 4.2: Context Model

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Context</strong></td>
<td></td>
</tr>
<tr>
<td>&quot;Language&quot;</td>
<td>Represents the language the user prefers.</td>
</tr>
<tr>
<td>&quot;FoodPreferences&quot;</td>
<td>Use to know the kind of food the user prefers.</td>
</tr>
<tr>
<td>&quot;Transport&quot;</td>
<td>Use to know the kind of transport the user prefers.</td>
</tr>
<tr>
<td>&quot;Impairments&quot;</td>
<td>Whether or not the user has physical impairments.</td>
</tr>
<tr>
<td>&quot;Location&quot;</td>
<td>Represents the geographic location of the user.</td>
</tr>
<tr>
<td><strong>Hardware Context</strong></td>
<td></td>
</tr>
<tr>
<td>&quot;DeviceID&quot;</td>
<td>Use to identify the mobile device.</td>
</tr>
<tr>
<td>&quot;CpuUsage&quot;</td>
<td>Represents the CPU usage of the mobile device.</td>
</tr>
<tr>
<td>&quot;EnergyLevel&quot;</td>
<td>Represents the energy level of the mobile device.</td>
</tr>
<tr>
<td>&quot;FreeMemory&quot;</td>
<td>Represents the available memory on the mobile device.</td>
</tr>
<tr>
<td>&quot;SignalStrength&quot;</td>
<td>Represents the strength of the connection signal of the mobile device.</td>
</tr>
</tbody>
</table>
Chapter 5

Implementation.

Hereafter are described details about the implementation of our approach, the description comprehends details about how the services, adaptations and the server are implemented, details about the implementation of mobile travel guide application are omitted because they are out of the scope of this work. However we point some references to the Mobile Travel Guide. This chapter is divided in two parts, the first one describes details about services and adaptations, and the latter deals with the server side of our application and data service providers.

5.1 Services and Adaptations.

The service holder and services, which are part of the client side of our approach, are implemented using the Extjs framework, they are implemented as extensions of the Ext.window and Ext.Panel classes provided by such framework. The implementation of the service holder is designed to fulfill the following requirements:

- Keep a registry of its inner components (services).
- Keep a registry of the expression (context information) a service needs.
- Handle requests and responses to and from Context Watcher (placed in the server side).
- Send notifications about context data to its inner components (services).

Figure 5.1 shows the structure for the classes we design for these purposes.

![Diagram of class structure](image)

**Figure 5.1: Client class structure implemented as Composite pattern.**

**Ext.ux.serviceHolder Class.**

It implements the before mentioned requirements, and also is the responsible for containing the services. It implements the user interface.

- `getContext` method. Handles requests to the Context Watcher on the server
side, it is in charge of set up a request (performing an RPC call). This method adds the following information to a request: a device identifier (the device we want to get context data from), the context data a service needs (expression), and a store object where data is saved after a response from the Context watcher is received.

- **getServiceNames and getSQLExpressions methods.** These methods get information about the inner components (services) of the container such as: Service identifiers and the expressions they have. Also they are used to create the expression registry.

- **loadEvent method.** Is an ad-hoc listener, when the store object is filled up with information from the Context Watcher this method is executed, and is responsible for forward the context data to a specific service.

- **deleteExpression and stilExp methods.** These are auxiliary methods which are used to delete expressions from the container registry, whenever an expression is satisfied and needs to be deleted from the service Holder.

These are the main methods the service holder class provides, in order to perform communication with the server side, this class implements the long polling approach, a request to the server is performed every time information from the Context watcher is received or a time-out occurs. The first RPC call obtains all procedures available in the server, with this information a JavaScript RPC object is created. The first RPC call is performed when an instance of the service holder is created, this call creates an instance of the context model for this client, after the context model is created, the service holder performs a new RPC call requesting the context data from the mobile device, according to the expressions submitted by a service.

*Ext.ux.servicePanel.*

The service Panel class is used to implement the different services and the adaptations rules that each of them executes. The main methods to perform its tasks are...
described below:

- addExpressions method. This method gives the possibility to add expressions as well as adaptation rules for that expression. To use this method is only necessary to write the expression and the rule which will perform the adaptation. It is possible to add and array of expressions or rules or both at once. When one expression and an array of rules are specified, the method will assign the same expression to all rules within the array, the same occurs when many expressions and one rule are added. However if an array of expressions and an array of rules are added, the length of both must be the same, because the method assumes there is one expression for each rule, is the lengths of both arrays are different an exception will be raised.

- setExpressions method. It is used by addExpressions method as an auxiliary to set up the expressions and rules within the service panel.

- getExpressions and getServiceId methods. They were designed to provide information about the service panel to an external container, in this case a container which is an instance of Ext.ux.serviceHolder class.

Adaptations Rules.

As stated before depending on context data either user data context, or hardware context data a service panel is able to perform two kinds of adaptations: functional or layout. Here is described how this kind of adaptations is performed according to its implementation.

The adaptations rules are part of a service they are defined as functions within the service. To define an adaptation rule is necessary to specify which components of the service are going to be adapted and what adaptation is going to be performed over the components. When adaptation rule is added to the service, as explained before, an expression (SQL-like sentence) and an adaptation rule need to be specified. This adaptation rule is executed when its expression is satisfied, because internally when
the addExpressions method from the service panel class is invoked, this method adds this expression as an event to the service and the adaptation rule as the function which will be performed when such event is triggered.

Figures 5.2 and 5.3 describe the required sequence in order to perform an adaptation. Hereafter, we describe the steps of such sequence, two main parts are consider: Subscription for context information and Context information handling.

Figure 5.2: Context Information Subscription.

To make a subscription the following steps are performed:

- The Service Holder obtains all the Expressions from all Services.
- The Service Holder sends Service - Expression value pairs to the Context Watcher.
- The Context Watcher stores the Service - Expression value pairs in the Service Expression Registry.
- The Context Watcher subscribes all the Expressions contained in the Service Expression Registry with Context Service.
- The Context Service generates requests either to MoCA web service or CroCo.
When an Expression is met, the following steps are performed:

- Either MoCA web service or CroCo retrieves the context information to the Context Service.
- The Context service retrieves the expression which has been met to the Context Watcher.
- The Context Watcher searches the received Expression in its Service Expression Registry and obtains the Service the Expression belongs to.
- The Context Watcher sends a Service - Expression value pair to the Service Holder.
- The Service Holder forwards the Expression to the Service specified by the received Service - Expression value pair.
- When a Service obtains the Expression the Adaptations are performed as already explained.

Examples about the adaptations are shown in the following images.
Figure 5.4: Layout adaptation based on language preference.

Figure 5.5: Place selection based on user location.
5.2 User profiles.

Our implementation provides predefined user profiles, which offers distinct services to the user. When the application is initialized, the user should choose one of them. We have defined three profiles, which are described below:

*Sport profile.*

It provides the following services: ”Weather forecast”, ”The Closest Place” and ”I am in a hurry”.

*Business profile.*

It provides the following services: ”Automatic Route Calculation for Agenda Appointments”, ”Weather forecast” and ”Personalized News”.

*Tourist profile.*

It provides the following services: ”Weather forecast”, ”The Closest Place”, ”Cheap Site Seeing”, ”Automatic Ticket Buying” and ”Special Eating”.

Figure 5.6: Layout adaptation based on signal strength.
5.3 Server side and Context Providers.

The server side is implemented using Java programming language, it consist of a main Java Servlet\(^1\), which contains instances of the Java classes that implement the described architecture. This main servlet is designed to be used by the Mobile Travel guide application, hence, new functionality we add more functionalities to fulfill our approach requirements. Functionalities such as: device authentication, context information handling, web service retrieval information and the how are implemented are here described.

*Context creation.*

Before the interaction between the client and the context watcher starts, a context model is created, i.e. the first request from the client (executed when the service holder is instantiated) creates the context model for the client. This is performed in the following way, the Context Watcher forwards the request to the Context service when the Context Service receives the request, it performs the corresponding requests to MoCA and CroCo context providers to create an instance of the context model, such requests are performed using the SOAP protocol, java stub classes are used to handle the communication between our java server and the context provider web services.

5.3.1 Context Watcher and Context Service.

As stated before, the Context Watcher contains an registry of Expressions and Services, this registry is used to track which expressions have been satisfied. After creating the registry the Context Watcher generates a request to the Context Service for each expression in the registry and waits for the response, when a Expression from the client is received by the Context Watcher, this is compared against the Expressions in the registry, if the Expression is already in the registry it is discharge and

\(^1\)A Servlet is a Java class which conforms to the Java Servlet API, a protocol by which a Java class may respond to HTTP requests.
no request to the Context Service is generated, only the Service is added. Otherwise, the Expression and the Service are stored and a new request to the Context Service is performed.

The service expression registry is implemented as a Hash Table, which contains an expression as key and services as values for that key. When a expression is met, the corresponding key is deleted from the Hash Table.

The Context Service is in charge of forwarding request from the Context Watcher to MoCA and CroCo context providers. It subscribes to the MoCA context provider sending SQL-expressions as parameters. It also request context information from the user to CroCo Web Service. This kind of request is performed sending SPARQL sentences as parameters, these sentences specify the required information from the context ontology modeled in CroCo.

Information about the current state of the mobile device (hardware context) is gathered running the MoCA monitor on the mobile device, this monitor runs as a daemon on the device and periodically send information to the CIS server. The MoCA web service accesses the context information gathered by the CIS server to provide our application with such information. The MoCA context provider was modified in order to adapt it to our requirements. The requestCISExp method was added to MoCA web service interface, this method gets a list with expressions, and returns the expression which has been met, according the CIS server, with this is possible to get a specific expression which is actually the information our architecture needs to work.

5.4 Web Service Information Retriever.

In Figure 5.7 is shown the way that service provider classes are implemented. Each class correspond to a service provided by the application. They implement the interfaces that the services access. The main method of these classes is the getService
method, which should be implemented according to each data service provider. All these classes and interfaces form the Web Service Information Retriever part of our application.

Figure 5.7: Service class diagram.
Chapter 6

Evaluation

In this chapter we describe the tests that were done in order to evaluate our approach as well as the results we obtained from such test. The chapter is structured as follows: first a description of the environment where the tests were performed is given, then a description of the tests and the variables that were taken into consideration is given, finally the results of the tests are presented.

6.1 Test environment.

Here a description of the server and clients are presented. All test were executed under the same environment, only some parameters in the configuration of the client monitors were modified according the requirements of each test, i.e. we modified the context information the monitors provide, in order to simulate different conditions on the state of the mobile device.

6.1.1 Server.

The server part of our approach was installed in a PowerBook G4 computer, running a 1.5 Java version (Java SE 5.0), with the following hardware characteristics:

• Processor: PowerPC G4 (1.2).

• Processor speed: 1.67 GHz.

• Memory: 512 MB

• OS Name: Mac OSX.

• OS version: 10.5.8.

We use the Apache Tomcat \(^1\) Servlet/JSP container, as the main container of our application. The version 6.0.20 was installed. The Tomcat container deploys all other applications which are needed to execute our application. Applications such as:

• Mobile Collaboration Architecture (MoCA) Web Service. Provides a language independent interface to MoCA architecture.

• CroCo Context Provider. Provides context information management to our application.

• Travelguide application. Application where our approach is implemented.

• Apache Derby Network Server. Relational Database management system running as embedded database providing support for client login.

Additionally to the already mentioned applications, the MoCA Context Information Service (CIS) was also installed on the computer. The CIS provides context information about the current state of a mobile device and this information is accessed by the MoCA Web Service.

\(^1\)http://tomcat.apache.org/
6.1.2 Monitors.

Monitor simulators were executing on client machines in order to simulate different state conditions of a mobile device (each monitor represents a mobile device), the attributes that were taken into consideration for the test were:

- The quality of the wireless connection.
- The remaining energy of the mobile device.
- The current CPU usage of the mobile device and,
- its remaining free memory.

The values of these four attributes were modifying while the test were executed, this to simulate changes on the current state of the mobile device. Also the following monitor configuration attributes were modified

- monitor.scanInterval. Periodicity (in milliseconds) of sending information to CIS service.
- fileX. File that have scans samples.
- and intervalX. Interval (in milliseconds) of sending information for fileX.

With the modification of these parameters we modified the period in which information about the state of the mobile device is sent to the CIS server and also the information we sent. An example of the information sent to the CIS server is presented in figure 6.1.

A web browser running on the client is also required. The tests were performed using the following web browsers: Firefox 3.5 and 3.6, Opera and Opera mobile 10.
6.2 Test Descriptions.

Within this section we describe the performed tests and their use cases, and the context variables we focused on. One test per profile was executed. With the tests we tried to check the following issues:

- Whether an adaptation on a service is performed when certain context information arrives or not.
- Whether the service presents errors after or during the adaptation.
- Server response times for service requests.

6.2.1 General Use Case.

The user accesses our application from his mobile device, after that he introduces the required information (device id, desired profile, personal preferences) in the configuration screen, clicks on "Apply" button to submit the information then on the "Services" button in order to see the services provided by the selected profile.

After the services screen is presented to the user, he starts examining the services and observes the adaptations (if any) on the services. The user finishes the interaction with the application once he has read the presented information.
6.2.2 Test 01: Sport profile user.

This test focuses on the creation of a new user using the provided Sports profile and on the usage of the services provided by such profile. As described before, our intention is to test if the adaptations are performed in a correct way. Also we measured the time elapsed between the arrival of a context event (context information about the mobile device) to our application server and the response of our server to the client for such event.

Considered context variable:

- Location. User current location.

This test was focused on the user location, therefore changes on location of the mobile device where simulated with the monitor.

6.2.3 Test 02: Business profile user.

This test focuses on the creation of a new user using the provided Business profile and on the usage of the services provided by such profile. As described before, our intention is to test if the adaptations are performed in a correct way. Also we measured the time elapsed between the arrival of a context event (context information about the mobile device) to our application server and the response of our server to the client for such event.

Considered context variables:

- Location. User current location.

- Language. The language user preference was modified in order to present same information in distinct languages.

- Food. The food preference was modified in order to retrieve information about restaurants based on the type of food and user current location.
6.2.4 Test 03: Tourist profile user.

This test focuses on the creation of a new user using the provided Tourist profile and on the usage of the services provided by such profile. As described before, our intention is to test if the adaptations are performed in a correct way. Also we measured the time elapsed between the arrival of a context event (context information about the mobile device) to our application server and the response of our server to the client for such event.

Considered context variables:

- Location. User current location.
- Signal strength. Changes on the signal strength were simulated to observe the information presented to the user.
- CPU and Memory usage. Changes to the values of these attributes were done in order to stop some of the services that were running in the profile.

6.3 Results.

This section presents the results obtained from the already described test. First we present the average response times of the server depending on the profile and number of concurrent users, after that we discuss about the tests and the problems presented during their execution.

Test 01: Sport profile user.

Maximum number of users: 8.

Provided Services:

"Weather forecast", "The Closest Place" and "I am in a hurry".

Graph 6.2 shows the time elapsed between the arrival of a context event and the response of our server to the client.
**Test 02: Business profile user.**

Maximum number of users: 8.

Provided Services:

"Automatic Route Calculation for Agenda Appointments", "Weather forecast" and "Personalized News".

Graph 6.3 shows the time elapsed between the arrival of a context event and the response of our server to the client.
Test 03: Tourist profile user.

Figure 6.4: Tourist profile.

Maximum number of users: 8.

Provided Services:

"Weather forecast", "The Closest Place", "Cheap Site Seeing", "Automatic Ticket Buying" and "Special Eating".

Graph 6.4 shows the elapsed time between the arrival of a context event and the response of our server to the client.

Discussion.

In general the evaluated services were able to perform a modification to themselves when information about the context of the device was received, these modifications did not cause errors in the service while executing and after the execution of the modifications, the service worked normally; only a short pause on the service was detected when executing layout adaptations, this is due to a layout adaptation, the service should render itself again, so the service was paused during the rendering.

Another problem detected during the execution of these tests was about location information and services that involve maps. Whenever information about the current location (coordinates) of the user arrives, this is received by the service and presented
to the user in a map; from this map the service can obtain more exact information about the location such as country, city, and street. With these data sets the service can request some other information depending on the service. The problem was that when a new location is set on a map, the map internally makes a request to the google maps api, if this request fails or the coordinates the service provide are not exact (from the point of view of google maps), one of two outcomes result: the map is undefined or no information about the current location is provided, these results cause an error in the service, because it cannot get the information it requires for its function. An attempt was made to solve the problem by stopping the service when this kind of problem arises, so the service is not available anymore.

We can also observe that the time elapsed between the arrival of a context event to our application server and the response of our server to the client for such event, increases when the number of users increases.
Chapter 7

Conclusion

Our approach presents in this work an architecture to perform adaptations (functional and user interface) based on context. In our approach, we obtained the following results: adapt user interfaces which are able to take into consideration context information about the user and the mobile device in order to perform adaptations on behalf of the user. Such services are able to react to changes in the context information modifying themselves either in their internal behavior or in their layout or both.

The UI services provide to the user different information, which is relevant to the user according his preferences, such as: weather information or appointment information.

To perform such modifications, the UI services should define adaptations rules. Two kinds of adaptation rules are defined for a service: functional adaptation rule and layout adaptation rule. The UI services provide support to define such adaptation rules within the service, adaptation rules are added as part of the service itself. To define both kinds of adaptation rules some elements are needed, like:

- An Expression which expresses the situation by which an adaptation is performed.
- The service components which are going to be adapted.
- The modification to be performed over a service when an expression is satisfied.
An architecture, which provides support for this UI services, was also designed. The architecture provides basic interfaces to query context information from different context providers, as well as for context profile creation. The context providers our architecture encloses are the Mobile Collaboration Architecture (MoCA) and the Ontology-Based, Cross-Application Context Management (CroCo). The architecture also provides access to different web services, which provides information to a UI service. The information a web service provides is specific to a service, this information can be about weather, information about places and appointments, for example. A context model based on the CroCo and MoCA context model was also developed, this context model encloses information about the user and the mobile device, the context variables our context model takes into consideration are: Language, FoodPreferences, Transport, Impairments, Location, DeviceID, CpuUsage, EnergyLevel, FreeMemory and SignalStrength. We consider these attributes as relevant to our approach. The architecture we present is the main result of our approach, and our main contribution focuses on the definition of context based adaptations of user interface or functional.
Bibliography


