Developing Unobtrusive Mobile Interactions

A Model Driven Engineering approach

Miriam Gil Pascual

UNIVERSIDAD POLITECNICA DE VALENCIA

Supervisors:
Dr. Vicente Pelechano Ferragud
Dr. Pau Giner Blasco

Centro de Investigación en
Métodos de Producción de Software

Miriam Gil

Developing Unobtrusive Mobile Interactions

A Model Driven Engineering approach

Master's Thesis, December 2010

PROS
Centro de Investigación en Métodos de Producción de Software

UNIVERSIDAD POLITECNICA DE VALENCIA

Developing Unobtrusive Mobile Interactions: A Model Driven Engineering approach

This report was prepared by
Miriam Gil

Supervisors
Vicente Pelechano
Pau Giner

Centro de Investigación en Métodos de Producción de Software
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia
Spain

Tel: (+34) 963 877 007 (Ext. 83530)
Fax: (+34) 963 877 359
Web: http://www.pros.upv.es

Release date: 10-12-2010
Comments: A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at the Universidad Politécnica de Valencia.

To my grandparents.

"A picture is worth a thousand words. An interface is worth a thousand pictures".

- Ben Shneiderman, 2003

Abstract

In Ubiquitous computing environments, people are surrounded by a lot of embedded services. With the inclusion of pervasive technologies such as sensors or GPS receivers, mobile devices turn into an effective communication tool between users and the services embedded in their environment. All these services compete for the attentional resources of the user. Thus, it is essential to consider the degree in which each service intrudes the user mind when services are designed.

In order to prevent service behavior from becoming overwhelming, this work, based on Model Driven Engineering foundations, is devoted to develop services according to user needs. In this thesis, we provide a systematic method for the development of mobile services that can be adapted in terms of obtrusiveness. That is, services can be developed to provide their functionality at different obtrusiveness levels by minimizing the duplication of efforts.

For the system specification, a modeling language is defined to cope with the particular requirements of the context-aware user interface domain. From this specification, following a sequence of well-defined steps, a software solution is obtained.

The proposal has been applied in practice with end-users. Although the development process is not completely automated, the guidance offered and the formalization of the involved concepts was proven helpful

to raise the abstraction level of development avoiding to deal with technological details.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Ubiquitous Computing vision implies a radical paradigm shift in the way users interact with systems (Weiser, 1991). One of the defining traits of this vision is the pursuit of **invisibility**. Computing resources become invisible to the user in order to allow interaction with the system in a *natural way*.

The ubiquity of mobile phones gives them great potential to be the default physical interface for ubiquitous computing applications (Ballagas et al., 2006). Furthermore, mobile devices have become full of sensors that provide a rich contextual information (e.g., current location of the user). However, this information is considered marginally in most of mobile services. This does not conform to our everyday life and behavior in which context plays a central role. Moreover, the uses of such pervasive technology can also disturb the users in the current focus of activity.

Mobile phones ring loudly even though we are in a meeting, computers interrupt presentations advising a software update. The infiltration of computer technologies into everyday life has brought these interac-

tion crises to a head. As Neis Gershenfeld observes, "There's a very real sense in which the things around us are infringing a new kind of right that has not needed protection until now. We're spending more and more time responding to the demands of machines." (Gershenfeld, 1999)

The challenge in an environment full of embedded services is to "provide the right information, at the right time and in the right way for individual users" (Fischer, 2001) avoiding to interrupting them. If we do not provide support to help control the interactions, users may be disturbed often, and be unable to focus on their tasks. This thesis provides an approach to define services that adapt their attentional demand according to the context of each user. In this work, modeling techniques are applied in order to face the development of such services from a higher abstraction level and systematize the construction of the final system.

The rest of this chapter is organized as follows: Section 1.1 explains the purpose of this work. Section 1.2 details the problem that the present thesis resolves. Section 1.3 introduces the goals defined for this work. Section 1.4 describes the approach followed in this thesis to fulfill the detected goals. Section 1.5 introduces the research methodology that has been followed in this work. Section 1.6 explains the context in which the work of this thesis has been performed. Finally, Section 1.7 gives an overview of the structure of this thesis.

1.1 Motivation

The pervasive computing paradigm envisions an environment full of embedded services. With the inclusion of pervasive technologies such as sensors or GPS receivers, mobile devices turn into an effective communication tool between users and the services embedded in their environment (e.g. users can compare prices of products directly interacting wih the physical products by means of the user mobile device).

Since mobile devices provide a rich contextual information about the user, the system can anticipate some of the user tasks. However a

complete automation is not always possible or desirable (Tedre, 2008). For certain tasks, some users prefer that the system acts silently in order not to be disturbed. For other tasks, users want to know what is happening behind the scenes. For example, when the favorite program of a certain user begins, the system should consider whether to start recording and/or informing the user depending on his/her context or preferences. If the system decides to inform the user first, it must choose the most adequate mechanism from all the ones available in his/her mobile device (sound, vibration, a text message, etc.).

Services should interact with users in a way that is not disturbing for them. Services might be embedded in the actual activities of everyday life, resulting in "calm" technology that moves back and forth between the center and the periphery of human attention (Weiser & Brown, 1997). Since user attention is a valuable but limited resource, an environment full of embedded services must behave in a considerate manner (Gibbs, 2004), demanding user attention only when it is actually required. The work of Presto (Giner et al., 2010) (a context-aware mobile platform that allows to support different workflows by interacting with the physical environment) highlighted the use of different levels of obtrusiveness to support workflow tasks. However, in many cases, the adequate intrusion level for a given task depends on the context. In the present work, we follow the idea of Presto in order to allow the degree to which interaction intrudes on user attention can be adapted according to context information and user needs.

Towards creating interfaces that adapt their level of intrusiveness to the context of use, several initiatives have studied strategies to minimize the burden on interruptions (Ho & Intille, 2005; Vastenburg et al., 2008). However, these initiatives are almost exclusively focused on evaluating the adequate timing for interruptions, while user interface adaptation has received few attention. Other initiatives that focus their works on development context-aware user interfaces (Calvary et al., 2003; Van den Bergh & Coninx, 2005) do not consider user attention in the adaptation process.

In an environment where the possible combinations of context are constantly increasing, the implementation of ad-hoc solutions to cover

all possible combinations is not feasible. Therefore, there is a need for a systematic development method of mobile services that can be adapted to regulate the service obtrusiveness (i.e., the extent to which each service intrudes the user's mind) without duplicating efforts in the development. Through decomposing the context aspects in its commonalities and differences, the interaction could be adapted to the user attention without explicitly having to define it for each combination of context. For example, a noisy context and a user with an auditory impairment require interaction not to be provided by means of audio. By considering the specification in terms of features (as opposed to specifying for each context), the duplication of efforts in the development are minimized since both cases are expressed as the exclusion of the auditory feature.

1.2 Problem statement

The development of context-aware user interfaces is not a closed research topic. The above discussion indicates that some problems still need to be considered. The work that has been done in this thesis seeks to improve the development of context-aware mobile services focusing on the adaptation to regulate the user attention by considering these problems, which can be stated by the following three research questions:

Research question 1. How should interaction be adapted in terms of attentional demand according to the context of use?

Research question 2. How to decompose the interaction in different adaptation aspects in order to avoid duplicating efforts in the development?

Research question 3. How can adaptation specifications be represented in models to develop context-aware user interfaces systematically?

These research questions are analyzed and answered in the following sections.

1.3 Thesis goals

The main goal of this thesis is to define a method for the development of mobile services that can be adapted in terms of obtrusiveness (i.e., the extend to which each service intrudes the user's mind).

First of all, regarding **research question 1**, one of the main goals of this work is the study of the interaction techniques and the parameters on which interaction will be adapted. Traditional human-computer interactions have focused on the realm of *explicit interaction*, where the use of computers rely in a command-based or graphical user interface. *Implicit interactions*, on the other hand, enable communication and action without explicit input or output. In the present work, user attention is the focus on design the adaptation since this is a limited resource that must be conserved. Providing a great degree of automation for services is not always the best option in terms of user satisfaction (Tedre, 2008). For some tasks, users prefer that the system acts silently in order not to be disturbed. For other tasks, users want to know what is happening and interact with the system. In order to manage the intrusive nature of interactions and clasify the services, we make use of the framework for implicit interaction presented by Ju and Leifer (Ju & Leifer, 2008).

Regarding **research question 2**, another goal of this work is to avoid duplicating efforts in the development defining the adaptation process in a declarative manner. Variability modeling has proven useful in mass-production domains to maximize reuse when developing a set of similar software systems (a system family) (Coplien et al., 1998). Because common system parts are clearly identified, it is possible to detect reuse opportunities for each new development. They are used in (Cetina et al., 2009) to describe an adaptation space of the system at service level. Thus, variability modeling techniques are used in this work for selecting different alternatives for presenting pervasive services to the user depending on him/her preferences, needs and context in terms of obtrusiveness. Interaction requirements are expressed by means of features and this work provides a mechanism to make use of the adequate interaction technique to provide the functionality according to the ob-

trusiveness level.

Regarding **research question 3**, one of the goals of the present work is to minimize the error-proneness of this kind of developments. In order to do so, a systematic method is defined to obtain the appropiate interaction from specifications by following a sequence of well-defined steps.

1.4 The proposed solution

Model Driven Engineering (MDE) (Schmidt, 2006) proposes the use of models as the basis for system development. A model is a simplification of a system, built with an intended goal in mind, that should be able to answer questions in place of the actual system (Bézivin & Gerbé, 2001). The use of models (such as model of planes in a wind tunnel or models of software systems) in engineering has a twofold benefit. On the one hand, models **guide the development** of a system. On the other hand, models allow to **reason about the system** avoiding to deal with technical details.

In a context where the possible combinations of users, situations and devices are constantly increasing, the implementation of ad-hoc solutions to cover all possible combinations is not feasible. Sottet in (Sottet et al., 2005) reports this problem and stresses the relevance of MDE for the modeling of interaction in Ubicomp systems. The specified system can be automatically generated from an abstract description. This approach addresses the technological heterogeneity problems found at the implementation level, which constitutes a need for interaction when multiple platforms and interaction modes are considered. In this work, we introduce a development process based on the foundations of MDE to specify the interaction aspects of Ubicomp systems. Specifically, this development process provides the following contributions:

A **design method** has been defined in order to capture the adaptation aspects for mobile interaction. The defined method enables designers to specify to which degree pervasive services must intrude the users mind and the way in which a user can interact with the mobile

device.

An **evaluation method** is provided in order to simulate the adaptation of mobile interaction in terms of obtrusiveness. This enables to anticipate adaptation results without actually implementing the supporting system. The method is fast to apply and it allows to reproduce a level of user experience that is considered to be very close to what users expect from a final system. The method allows to receive relevant feedback in terms of user experience and interaction adaptation, which is essential to reconsider the designs prior to the development of the final system.

A **development method** is defined to guide the developer in the construction of the context-aware user interfaces in a systematic way. The method comprises from specification to the final implementation.

1.5 Research methodology

In order to perform the work of this thesis, we have carried out a research project following the design methodology for performing research in information systems as described by (March & Smith, 1995) and (Vaishnavi & Kuechler, 2004). Design research involves the analysis of the use and performance of designed artifacts to understand, explain and, very frequently, to improve on the behavior of aspects of Information Systems (Vaishnavi & Kuechler, 2004).

The design cycle consists of 5 process steps: (1) awareness of the problem, (2) solution suggestion, (3) design and development, (4) evaluation, and (5) conclusion. The design cycle is an iterative process; knowledge produced in the process by constructing and evaluating new artifacts is used as input for a better awareness of the problem. Following the cycle defined in the design research methodology, we started with the awareness of the problem (see Fig. 1.1): we identified the problem to be resolved and we stated it clearly.

Next, we performed the second step which is comprised of the suggestion of a solution to the problem, and comparing the improvements that this solution introduces with already existing solutions. To do this,

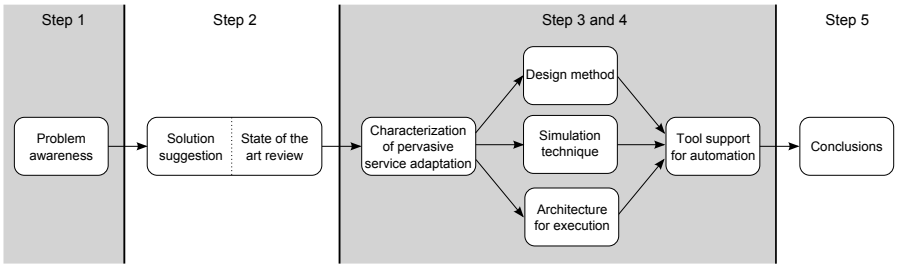

Figure 1.1: Research methodology followed in this thesis.

the most relevant approaches were studied in detail. Once the solution to the problem was described, we plan to develop and validate it (steps 3 and 4). These two steps are performed in several phases (see Fig. 1.1). The tasks carried out in these steps were intended to characterize pervasive service adaptation, define techniques for their design, simulation and implementation providing tool support for each.

Finally, we will analyze the results of our research work in order to obtain several conclusions as well as to delimitate areas for further research (step 5).

1.6 Thesis context

This Master's Thesis was developed in the context of the research center *Centro de Investigación en Métodos de Producción de Software* of the *Universidad Politécnica de Valencia*. The work that has made the development of this thesis possible is in the context of the following research government projects:

- SESAMO: Construcción de Servicios Software a partir de Modelos. CYCIT project referenced as TIN2007-62894.
- OSAMI Commons: Open Source Ambient Intelligence Commons. ITEA 2 project referenced as TSI-020400-2008-114.

- "Internet de las Cosas como soporte a Procesos de Negocio". Primeros proyectos de Investigación de la UPV, referenced as PAID-06-09 number 2920.

1.7 Thesis structure

This thesis is presented in seven chapters including this one. As a guide to the organization of the remainder of this thesis:

Chapter 2 introduces fields that are related to the work that is presented in this thesis and gives an overview of some relevant concepts and technologies in which this work relies.

Chapter 3 presents initiatives that face similar problems or provide solutions to some pieces of this work. This initiatives are analized and compared to the work that is presented in this thesis.

Chapter 4 defines the design method that is followed in our approach to adapt the way in which a pervasive service is accessed and describes the tools provided to design this kind of systems.

Chapter 5 provides guidelines to validate in practice the designs obtained with the method and introduces a development process that covers from the specification of the system to the implementation of the final solution by following a sequence of systematic steps. Code generation techniques are provided to obtain an implementation of the final system.

Chapter 6 details how the proposal has been validated by the development of a complete case study.

Chapter 7 summarizes the main contributions and publications of this work. In addition, this chapter provides some insights about further work.

Chapter 2

Background

This work deals with the development of mobile services that can be adapted in terms of obtrusiveness. As it is shown in Fig. 2.1, it is placed in the intersection of three research areas that have some aspects in common. This disciplines are: Mobile Computing, Context-Aware Computing and Considerate Computing.

This work relies on different concepts and technologies from these areas. In order to clarify the foundations in which our approach relies, different technologies and techniques are introduced in this chapter. The rest of this chapter is organized as follows: Section 2.1 provides an overview of the impact of context in the design and development of mobile systems and mobile platforms for coping with these requirements. Section 2.2 provides an overview of the context-aware area and the modeling languages that enable the design of context-aware user interfaces. Section 2.3 presents the foundations of the considerate computing paradigm for the development of user interfaces and introduces strategies to minimize interruptions. Finally, Section 2.4 concludes the chapter.

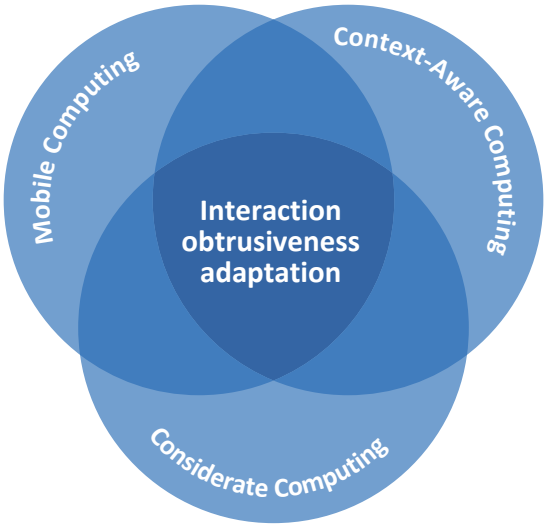

Figure 2.1: Application domains involved in this work.

2.1 Mobile Computing

Mark Weiser envisioned ubiquitous computing (Ubicomp) as a world where computation and communication would be conveniently at hand and distributed throughout our everyday environment (Weiser, 1991). Pervasive Computing (PerCom) (Hansmann et al., 2001), Ambient Intelligence (AmI) (Aarts et al., 2002) or Everyware (Greenfield, 2006) are some of the paradigms that share this goal. As mobile phones are rapidly becoming more powerful, this is beginning to become reality. Mobile devices play an important role in the modern society. With the advanced capabilities of mobile devices such as connectivity, positioning systems, sensors, and advanced interaction mechanisms, new valuable services can be provided. They are considered the first pervasively available computer and interaction device.

However, mobile interaction has peculiar aspects that distinguishes it from interaction with desktop systems, making it more difficult to build effective user interfaces for mobile users. One of the themes of this

work is to provide interaction with the system in a *natural way* through adapting interaction mechanisms to the different contexts. The way users interact with mobile devices is a key point to achieve a seamless and intuitive interaction. The different kinds of interaction are introduced below.

2.1.1 Interaction Modalities

Broadly speaking, there are two extremes of interaction: one in which the user interacts consciously and explicitly with the system; and at the other extreme, the user interacts unconsciously or implicitly (Fig. 2.2). In between, there are various degrees of each kind.

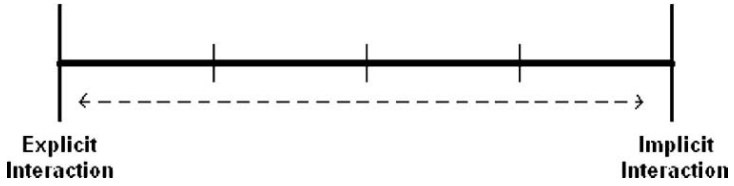

Figure 2.2: The interaction continuum (O'Grady et al., 2008)

Explicit Interaction In this case, a user interacts with a software application directly by manipulating a GUI, running a command in a command window or issuing a voice command. In short, the user intentionally performs some action.

Implicit Interaction Implicit interaction (Schmidt, 2000) is a more recent development and occurs when a user's subconscious actions indicate a preference that may be interpreted as an interaction. It is difficult to capture and subject to error, but it offers useful possibilities in the mobile domain. As a practical example, consider the case of a smart refrigerator. The refrigerator is aware of what products are put inside it. By not returning a product inside could be interpreted by a refrigerator as an indication of a lack of the product. Therefore, the refrigerator could deduce that

the user runs out of that product and it could add the product to the shopping list.

Implicit interaction is closely related to user's context (Tamminen et al., 2004) and some knowledge of the prevailing context is almost essential if designers want to incorporate it into their applications (O'Grady et al., 2008). Mobile information systems users are characterized by frequent changes in the context (Siau, 2003). Mobility emphasizes several concerns (space, time, personality, society, environment, an so on) often not considered by the traditional desktop systems (Krogstie et al., 2004). In addition, many limitations in terms of computing capabilities, screen size and so on, must be considered when systems are designed for being accessed through a mobile device. In particular, this work is interested in *environment context* (entities that surround the user) and *user context* (user using the mobile). The main challenges in the design of mobile applications are detailed below.

2.1.2 Challenges in mobile design

Mobile devices present designers with five main challenges (Dunlop & Brewster, 2002):

1. *Designing for mobility:* as users are mobile, their environment changes drastically as the user moves, varying available resources and application requirements.

2. *Designing for a widespread population:* mobile devices are typically used by a larger population spread than traditional PCs and without any training.

3. *Designing for limited input/output facilities:* screens are small due to the need for portability. Audio output quality is often very poor with restricted voice recognition on input. Keyboards are limited in size and number of keys.

4. *Designing for (incomplete and varying) context information:* mobile devices can be made aware of their context (e.g. current

location through the GPS).

5. *Designing for users multitasking at levels unfamiliar to most desktop users:* interruptions in mobile devices are likely to be much higher.

The development of a mobile application should take into account all of these kind of devices and situations. Software development facilities must be provided for leveraging the great capabilities of mobile devices. However, this has not been an easy task. Mobile operating systems have been mainly closed to developers. While a few have opened up to the point where they will allow some Java-based applications to run within a small environment on the phone, many do not allow this (DiMarzio, 2008).

The Open Handset Alliance[1] (OHA) was formed by Google with the goal of providing the first open, complete, and free platform created specifically for mobile devices. Different organizations joined the OHA, resulting the Android platform. The first version of Android was released on November 2007, almost one year later (in October 2008) the first mobile device supporting Android was released. In 2010 Android was featured in 20 devices and many more were planned including mobile phones, laptops, digital picture frames, e-book readers, home appliances, and gaming devices among others.

The availability of an open platform for mobile devices enables the development of innovative applications. Considering the number of sensors that are present in current mobile devices, developing context-aware mobile applications is feasible with the Android platform. For example, Android devices are capable of determining context information such as the user location and orientation. In this work, the Android platform is used for the development of context-aware user interfaces. Next section provides more detail about this platform.

[1] http://www.openhandsetalliance.com

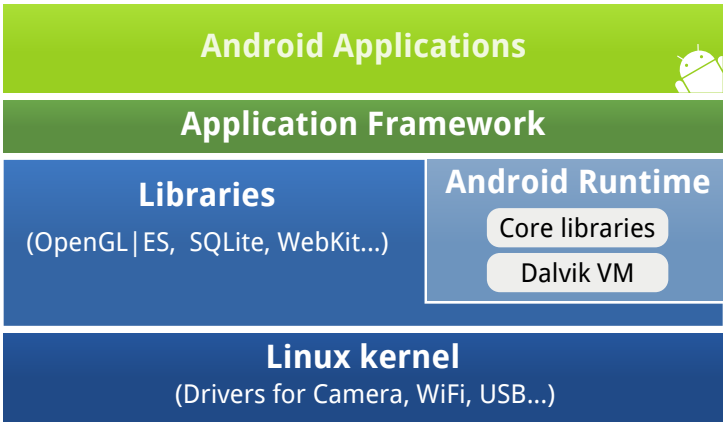

Figure 2.3: Android architecture

2.1.3 The Android platform

Android is an open software platform for mobile development that is intended to provide a full-stack for developers. Android includes an operating system, middleware, applications and development tools. This section describes the most innovative aspects of the Android platform since it has been used in this work.

Figure 2.3 provides an overview of the architecture of the platform. The platform is defined in different layers. Is worth noting that all the applications make use of the platform in the same way. There are not special privileges for certain applications (e.g., preinstalled applications) and any application can access to any platform service.

The Android Operating System is based on Linux 2.6. Mobile device manufacturers are in charge of developing drivers for their devices that follow the Linux directives. On top of the operating system, basic system functionality is provided by native C/C++ libraries. Most of them provide low-level functionality based on well established open source projects such as OpenGL|ES, FreeType, SQLite or WebKit. This layer abstracts the particularities of each hardware piece. For example, OpenGL can be used to program graphics regardless of the technology

used for the device screen.

One of the native components that plays a central role for the development of Android applications is the Dalvik Virtual Machine. Dalvik allows the development of applications for Android in Java. The Dalvik virtual machine allows the system functionality to be accessed from a higher abstraction layer. The Dalvik can run classes compiled by a Java language compiler that have been transformed into Dalvik Executable (*.dex) format, a format that is optimized for efficient storage and memory-mappable execution.

On top of the virtual machine Java utility functions based on Apache Harmony[2] are provided. These libraries provide interfaces and classes for programming. Data structures such as collections, connectivity functions to handle sockets and input and output access are only some of the functionalities that are provided with these core libraries. In order to facilitate the development an application framework is also provided. This framework provides the basic building blocks, communication mechanisms and APIs that any Android application will use. From the software engineering perspective, the application framework is the most relevant layer in the Android platform since it defined the main components that form any Android-based application. The main components that form the framework are detailed below.

2.1.4 The Android application framework

The Android application framework is based on loosely-coupled components. Each component developed is declared in the Android Manifest. When a component is described in the Android Manifest, it defines the way it is integrated in the platform by indicating the possibilities for communicating with other components and the permissions that each application requires for the platform. In this way, when an application is installed by users, they can know which kind of use of their mobile device would make a certain application.

The Android application framework provides the following main

[2]http://harmony.apache.org/

components: *Activity*, *Service*, *Content Provider* and *Broadcast Receiver*. These components are introduced below.

Activity. An Activity presents a visual user interface to the user. An activity is designed around a well-defined purpose (e.g., viewing, editing, dialing the phone, taking a photo, etc.). It handles a particular type of content (e.g., a list of contacts) and accepts a set of actions. The user interface provided by the Activity is composed by a hierarchy of interaction nodes (*View* and *ViewGroup* elements following the composite pattern) that process input events. Each activity has a lifecycle that is independent of the other activities in its application or task and it is managed by the application framework.

Service. A Service provides functionality that is executed in the background (e.g., a service that plays music). It is possible to connect to an ongoing service and start it if it is not already running. While connected, communication with the service is performed through an interface that the service exposes. Different components such as Activities or other Services can be binded to a Service.

Content Provider. A content provider makes data available to other applications. The data can be internally stored in the file system, in an SQLite database, or in any other particular mechanism. A Content Provider exposes generic mechanisms for accessing the information regardless of the underlying implementation technology used.

Broadcast Receiver. A broadcast receiver is a component that reacts to announcements from other components. Broadcasts can originate from system code (e.g., indicate that the battery is low) or other applications. In response to the broadcast, Broadcast Receivers can start an activity or use the *NotificationManager* to alert the user. Notifications can get the user's attention in various ways (flashing the backlight, vibrating the device, playing a sound, etc.).

Android allows different components to execute simultaneously and it provides an inter-component communication mechanism based on Intents. An *Intent* is an abstract description of a desired action (e.g., obtaining an image) regardless of the component that provides this functionality. Components declare in their manifest which kinds of intents they can respond to, and the linkage between the caller and the called is performed dynamically at run-time. This enables the extensibility of the platform since newly installed applications can take advantage of already installed components.

2.2 Context-Aware Computing

With the emergence of a wide variety of computer devices including mobile phones, Personal digital Assistants (PDAs), pocket PCs and so on, and the varying capabilities of these devices, it appeared the need to adapt this new diversity to the *context of use* without exploding the cost of development and maintenance. This is the goal of context-aware computing (Calvary et al., 2003). *Context-aware computing* as mentioned by Dey and Abowd (Dey & Abowd, 2000) refers to the "ability of computing devices to detect and sense, interpret and respond to, aspects of a user's local environment and the computing devices themselves". As context influences the interaction, the assumption is that by making systems aware of their context, interaction can be made more relevant and useful (Lucas, 2001).

In order to achieve the adaptation, devices, services or systems have access to context information. Context needs to be sensed and modeled to guide adaptations of interaction. There is no unequivocal definition of the concept of *context*. There are several definitions of *context* in the literature, but the most used definition in AmI systems was introduced by Dey (Dey & Abowd, 2000):

> "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user

and applications themselves."

There is a wide variety of information that can be gathered to support the adaptation process. However, most of the approaches define the context of use by three classes of entities (Calvary et al., 2003):

- *User*. The users of the system who are intended to use the system. In particular, his perceptual, cognitive and action disabilities.
- *Platform*. The computational and interaction device that can be used for interacting with the system. Examples, in terms of resources, include memory size, network bandwidth, screen size, input and output facilities, and so on.
- *Environment*. The physical environment where the interaction can take place such as noisy environment, lighting conditions, location information, time information, etc.

User interface modeling is an essential component of any effective long term approach to developing context-aware UIs. User interface modeling (Thevenin & Coutaz, 1999) involves the definition of several models pertaining to various facets of the UI, such as the presentation, the dialog, the platform, the user, the task, and the context. These models are then exploited to automatically produce a usable UI matching the requirements of each context of use. The present work deals with the modeling of interaction, taking into account its nomadic and context-aware nature. The current user interface modeling languages that allow the development of this kind of UIs are presented below.

2.2.1 Modeling languages

Different modeling languages are focused on the description of interaction. Developers can use a high-level language to implement an abstract and device-independent UI model. Then, they can generate the code for a specific platform (Seffah et al., 2004). Calvary et al. (Calvary et al., 2003) give an overview of different modeling approaches to deal with

user interfaces supporting multiple targets in the field of context-aware computing. The approaches studied from the user interface modeling area are detailed below.

User Interface Markup Language. The User Interface markup Language (UIML) (Abrams et al., 1999) is an XML-compliant language to support the development of UIs for multiple computing platforms by introducing a description that is platform-independent that will be further expanded with peers once a target platform has been chosen. Thus, the universality of UIML makes it possible to describe a rich set of interfaces and reduces the work in porting the user interface to another platform to changing the style description.

UIML describes a user interface with five sections: *description*, *structure*, *data*, *style*, and *events*. The *description* section lists the individual elements that collectively form an application's user interface. The *structure* section specifies which elements from the description section are present for a given appliance, and how the elements are organized. The *data* section contains data that is appliance-independent but application-dependent. The *style* section contains the style sheet information and data that are appliance-dependent. Finally, the *events* section describes the runtime interface events, which may be communicated.

UIML is a declarative language that distinguishes *which* user interface elements are present in an interface, *what* the structure of the elements are for a family of similar appliances, *what* natural language text should be used with the interface, *how* the interface is to be presented or rendered using cascading style sheets, and *how* events are to be handled for each user interface element.

eXtensible Interface Markup Language. eXtensible Interface Markup Language (XIML) (Puerta & Eisenstein, 2002) is an XML-based language that enables a framework for the definition and interrelation of interaction data items. XIML is an organized collection of interface elements that are categorized into one or more major

interface components. The language does not limit the number and types of components that can be defined.

In its first version, XIML predefines five basic interface components, namely *task*, *domain*, *user*, *dialog*, and *presentation*. The first three of these can be characterized as contextual and abstract, while the last two can be described as implementation-based and concrete. An XIML-based UI specification can lead to both an interpretation at runtime and a code generation at design time.

Using XIML, the design and implementation of a UI will be a series of refinements from an abstract UI representation (of the user context or a task model for example) to a concrete UI representation (of widgets and interaction techniques for example).

USer Interface eXtensible Markup Language. A User Interface Description language (USIXML) (Limbourg et al., 2004) aimed at describing user interfaces with various levels of details and abstractions, depending on the context of use. USIXML supports a family of user interfaces such as devide-independent, platform-independent, modality independent, and context-independent.

USIXML allows specifying multiple models involved in user interface design such as: task, domain, presentation, dialog, and context of use, which is in turn decomposed into user, platform, and environment.

It is focused on business applications and tries to be as complete as possible in the definition of all the models that are considered to be relevant. The language allows the specification of *domain*, *task*, *abstract user interface*, *concrete user interface* and *context models*. Mappings and transformations between the different models can be explicitly defined in separate models: mapping, transformation and rule-term model.

Context-Sensitive User Interface Profile. Bergh and Coninx introduced the Context-Sensitive User Interface Profile (CUP) (den Bergh & Coninx, 2005). CUP is a UML-based notation that allows the specification of requirements for context integration into

an interactive application. CUP proposes some improvements on UMLi (da Silva & Paton, 2003) to consider context aspects. CUP takes some notions from the Context Modeling language (Henricksen & Indulska, 2005) and defines a UML-based modeling language to define interactions based on them.

CUP is defined by means of the UML profile extension mechanism. The following models are defined: The *application model* shows the concepts and the relations between them that are used within the application. The *task dialog model* provides an hierarchical view to the activities that need to be accomplished. The *context model* shows the concepts that can influence the interaction of the user with the application directly or indirectly. The *abstract presentation model* shows the composition of interactors in the user interface and describes the general properties of the interactors (the data they interact with and meta information about them). *The concrete presentation model* describes the user interface for a specific set of contexts or platforms.

The context model defined considers the nature of gathering context information (manually entered into the system by the user/designer of the software or automatically sensed/interpreted), the information about the platform through which the user interacts with, context information ambiguity, and the topic of interest.

2.2.2 Analysis and discussion

Summarizing the results from context-aware computing, the goal of using context information is to make interaction with a system more relevant, useful and robust. Several conclusions arise from the analysis of the modeling languages studied.

Despite the evolution of the modeling languages for the designing of user interfaces, there is no accepted models and notations in contrast to the world of system design where the Unified Modeling Language (UML) (Rumbaugh et al., 1998) has emerged as the modeling notation of choice.

Several notations for model-based design of user interfaces, and more generally interactive systems, have been proposed. They all use several models to support the design process. The names, contents and number of these models vary greatly(den Bergh & Coninx, 2005). Despite of this, several models seem to come back in most approaches.

2.3 Considerate Computing

Humanity has connected itself through roughly three billion networked telephones, computers, traffic lights -even refrigerators and picture frames- because these things make life more comfortable and keep us available to those we care about. So although we could simply turn off the phones, close the e-mail program, and shut the office door when it is time for a meeting, we usually do not. We just endure the consequences(Gibbs, 2004).

Today, increasing numbers of users are surrounded by multiple ubiquitous computing devices, such as mobile phones, PDAs and so on. These devices bombard users with requests for attention, regardless of the cost of interruptions(Vertegaal, 2003). Numerous studies have shown that when people are unexpectedly interrupted, they not only work less efficiently but also make more mistakes. Eric Horvitz of Microsoft Research stated: "If we could just give our computers and phones some understanding of the limits of human attention and memory, it would make them seem a lot more thoughtful and courteous".

Researchers are becoming aware of the fact that user attention is a limited resource that must be conserved. To this end, the considerate computing paradigm aims at avoiding overloading the user by adapting system behavior based on the sensed user attention focus. Considerate user interfaces generally calculate the cost in terms of user attention and the benefit in terms of subjective or objective performance factors, in order to predict acceptability and select the optimal timing of the interruptions.

User interface designers and engineers should design computing devices that negotiate rather than impose the volume and timing of their

communications with the user. Cooper and Reimann's About Face 3 describes some of the most important characteristics of considerate interactive products:

> "Take an interest. Are deferential. Are forthcoming. Use common sense. Anticipate people's needs. Are conscientious. Don't burden you with their personal problems. Keep you informed. Are perceptive. Are self-confident. Don't ask a lot of questions. Take responsability. Know when to bend the rules." (Cooper et al., 2007)

Since user attention is a valuable but limited resource, an environment full of embedded services must behave in a considerate manner, demanding user attention only when it is actually required. The present work is interested in regulate the service obtrusiveness (i.e., the extent to which each service intrudes the user's mind). Several initiatives that have studied strategies to minimize the burden of interruptions are presented below.

BusyBody. Horvitz et al. described BusiBody (Horvitz et al., 2004), a software component that provides an integrated, onboard supervised learning and inference system. It is a system that intermittently asks users to assess their perceived interruptability during a training phase and that builds decision-theoretic models with the ability to predict the cost of interrupting the user.

BusyBody intermittently engages users via a pop-up *busy palette*, heralded with an audio chime. The palette allows users to assess their current cost of interruption efficiently. In the background, a rich stream of desktop events is logged continuously. BusyBody trains and periodically re-trains Bayesian network models that provide real-time inferences about the cost of notification. The models are linked to programming interfaces that allow other components, such as notification systems to access the expected cost of interruption.

Multi-Agent Negotiation. Ramchurn et al. proposed an agent-based approach to minimizing intrusiveness in a meeting environment (Ramchurn et al., 2004). It mainly concerns how to display a message when there is a meeting.

Participants have their own devices, such as a personal laptop; they also share some public devices, such as a whiteboard in the meeting room. Displaying a message on a public device is more intrusive than displaying it on a private device, as all participants can see the whiteboard but not the screen of a laptop. Each user has an agent maintaining the user's interests and making decisions for him. When a message arrives for a user, the agent checks with other agents to see if they are interested in this message. If so, the message will be displayed on a public device; otherwise, it goes to a private device.

AuraOrb. AuraOrb (Altosaar et al., 2006) is a social appliance, an ambient notification device that uses information about user interest to determine its notification strategy. AuraOrb uses eye contact sensing to detect user interest in an initially ambient light display. When eye contact is detected, AuraOrb displays the subject heading of the notification, i.e., in the center of the user's attention. Touching the orb causes the associated message to be displayed on the user's last attended computer screen. When user interest is lost, AuraOrb automatically reverts back to its idle state.

Considerate Home Notification Systems. Vastenburg et al. (Vastenburg et al., 2009) conducted a user study of acceptability of notifications in a living room laboratory in order to (1) know what factors influence the acceptability of notifications and (2) create a considerate mechanism for scheduling and presenting notifications in the home. The study was concentrated on a potentially relevant factor: the level of intrusiveness. Secondary factors included were: engagement in activities and message urgency.

The study was situated in a living room laboratory in which the user activities and the timing of notifications were controlled.

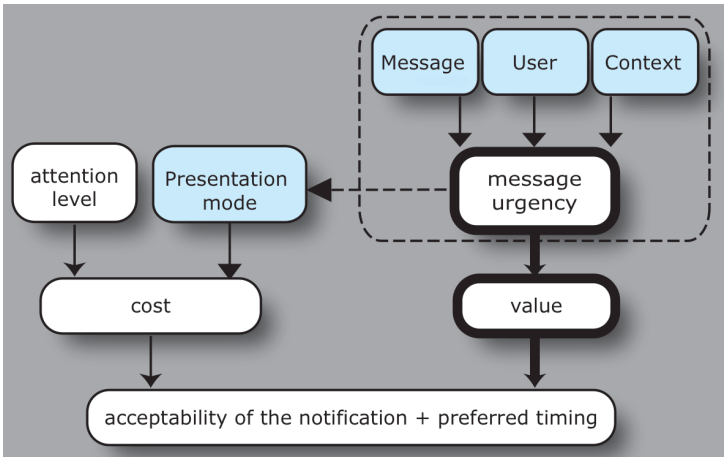

Figure 2.4: Model of acceptability of notifications (Vastenburg et al., 2009).

They evaluated questionnaire data using cluster analysis in order to construct a semantic model that describes the relationship between user, system and environment. This model is shown in Fig 2.4. The model shows that perceived cost of notifications is linked to attention level and presentation mode. Moreover, perceived value of messages is linked to perceived message urgency. Prediction of message urgency remains a major challenge in the development of considerate notification systems. The bold arrows indicate message urgency to be the primary indicator of acceptability and preferred timing.

2.3.1 Analysis and discussion

Several works were presented with the same goal: reduce the perceived burden of interruptions. Existing systems deal with the problem in different ways but with various limitations.

For example, the work of Ramchurn does not address the problem of whether an incoming message is intrusive to the receiver. Furthermore, it can only be applied to a situation where multiple users have shared

devices.

In the same manner, BusyBody and AuraOrb are unaware of the value of the messages itself, resulting in potentially unwanted interruptions in case of non-urgent messages.

The conclusion is that these initiatives are almost exclusively focused on evaluating the adequate timing for interruptions, while user interface adaptation has received few attention.

2.4 Conclusions

This chapter provides an overview of different techniques that are related with the work presented in this document. The analisys has considered three application domains: Mobile Computing, Context-Aware Computing and Considerate Computing. This work is aimed at providing development support for systems that fit in these three areas. Thus, much of the technologies and techniques introduced are applied in the following chapters. A more detailed analysis of the proposals that are more close to the goal of this work is provided in Chapter 3.

Chapter 3

State of the art

This chapter introduces the most important approaches that support the design and development of considerate mobile user interfaces. Once we have analyzed in Chapter 2 the general application domains in which this work fits, we analize the specific proposals in this domains that are closely related to our approach. This analysis allows us to determine the way in which each proposal addresses the aspects that are central in our approach.

The work of this thesis is placed in the intersection of *Mobile Computing*, *Context-Aware Computing* and *Considerate Computing*. An overview of the techniques used in these areas was presented in Chapter 2. We have identified three subdomains situated in the intersection of the main areas that are relevant to the present work. This is illustrated in the Figure 3.1. In particular, *context-aware mobile user interfaces* deals with the development of context-aware user interfaces in the domain of mobile computing, *non-intrusive mobile computing* is based on reducing the burden of interruptions in the mobile area, and finally, *attentive user interfaces* are context-aware user interfaces where

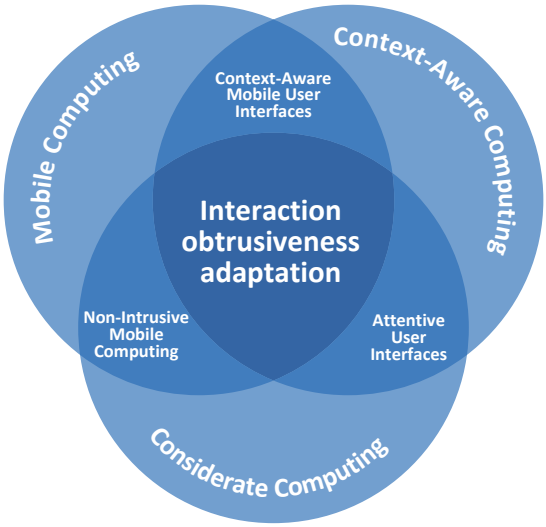

Figure 3.1: Application domains involved in this work and their intersecting subdomains.

context is governed by user attention. Relevant approaches in these subareas have been analyzed and discussed in this chapter.

The remainder of the chapter is structured as follows. Section 3.1 presents related work for adapting user interfaces to mobile contexts. Section 3.2 studies different approaches with the goal of reducing the intrusiveness in mobile computing. Section 3.3 introduces the research carried out in the *attentive user interfaces* area. Finally, Section 3.4 concludes the chapter.

3.1 Context-Aware Mobile User Interfaces

As mobile computing is growing and offering a variety of access devices for multiple contexts of use, it poses the need to design and develop context-aware user interfaces for mobile contexts. Some of the challenges introduced by mobile computing in the design and development of context-aware user interfaces are (Satyanarayanan, 1996): the diver-

sity of computing platforms, the input and output constraints for each platform, the contextual change while the user is carrying out the task, and the varying contexts for which mobile devices are suited. Relevant techniques and tools from this area are introduced below.

MANNA: The Map ANNotation Assistant. Eisenstein described MANNA (Eisenstein et al., 2001), a hypothetical software application intended to create annotated maps of geographical areas. It is a multimedia application that must run on several platforms and can be utilized collaboratively over the internet. In order to provide an adaptive, multi-platform user interface, it describes a set of model-based techniques that can facilitate the design of such UIs.

The technique used is based on the development of a user interface model. A UI model is expressed by the MIMIC modeling language (Puerta, 1996). MIMIC is a comprehensive UI modeling language which include all relevant aspects of the UI. In particular, it focused on three model components: *platform model* which describes the various computer systems that may run a UI, *presentation model* that describes the visual appearance of the user interface, and *task model* that is a structured representation of the tasks that the user may want to perform.

The connections between the platform model and the presentation model describe how the constraints posed by the various platforms will influence the visual UI appearance. Through the application of a task model, it can optimize the UI for each device since, depending on the device, the user may want to perform a subset of the overall tasks. The overview of the procedure is depicted in Fig. 3.2.

The Unifying Reference Framework. Calvary et al. (Calvary et al., 2003) describe a development process to create context-aware user interfaces. It was presented to characterize the models, the methods, and the process involved for developing user interfaces for multiple contexts of use. In this framework, a context of use is

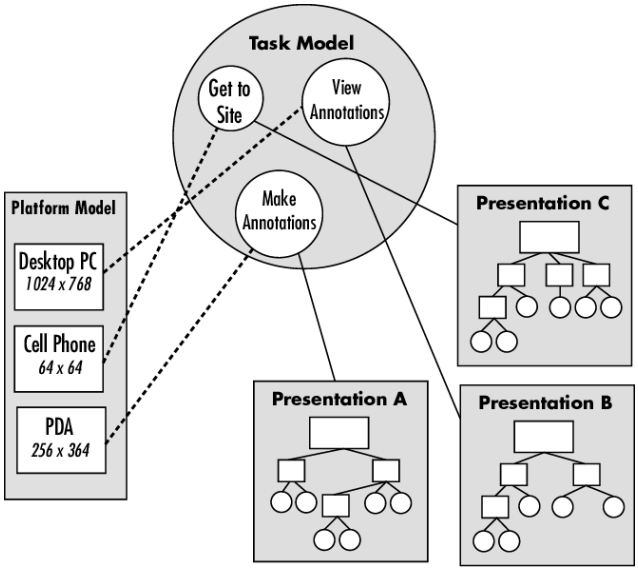

Figure 3.2: Overview of the approach followed by MANNA (Eisenstein et al., 2001).

decomposed into three facets: the *end users* of the interactive system, the *hardware and software computing platform* with which the users have to carry out their interactive tasks and the *physical environment* where they are working.

The development process consists of four steps: creation of a task-oriented specification, creation of the abstract interface, creation of the concrete interface, and finally the creation of the context-aware interactive system. These levels are structured with a relationship of reification going from an abstract level to a concrete one and a relationship of abstraction going from a concrete level to an abstract one. The focus however, lays upon a mechanism for context detection and how context information can be used to adapt the UI, captured in a three-step process: (1) recognizing the current situation (2) calculating the reaction and (3) executing the reaction.

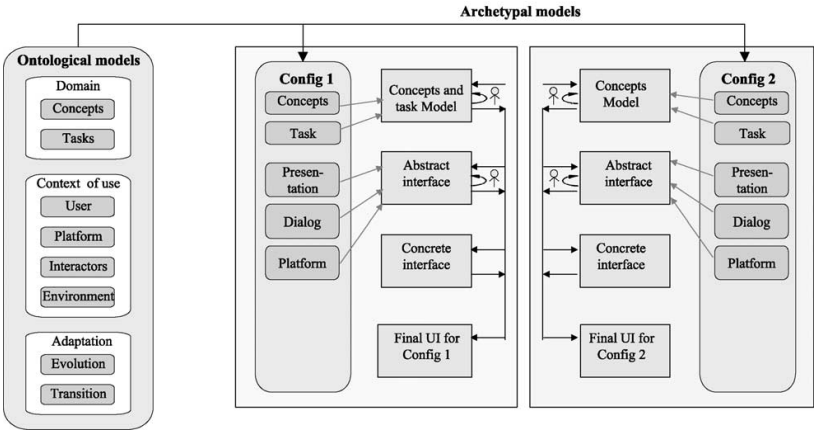

Figure 3.3: The unifying reference framework instantiated for TERESA (Calvary et al., 2003).

TERESA. TERESA (Transformation Environment for inteRactivE Systems representAtions) (Mori et al., 2004) is a tool for designing user interfaces for various platforms including mobile devices. TERESA presents a solucion based on three levels of abstractions. These levels involved are: The *task and object model* which considers the logical activities that need to be performed, the *abstract user interface* defined as a number of abstract presentations, a *concrete user interface* where each abstract interactor is replaced with a concrete interaction object, and the *final user interface* which is the result of the translation of the concrete interface into an specific software language.

TERESA exploits a UIDL called TERESAXML that supports several types of transformations such as: task model into presentation task sets, task model into abstract UI, abstract UI to concrete UI, and generation of the final UI. Figure 3.3 graphically depicts the Unifying Reference Framework (Calvary et al., 2003) instantiated for the method described.

Cameleon RT. Cameleon RT (Balme et al., 2004) is a reference model constructed to define the problem space of user interfaces re-

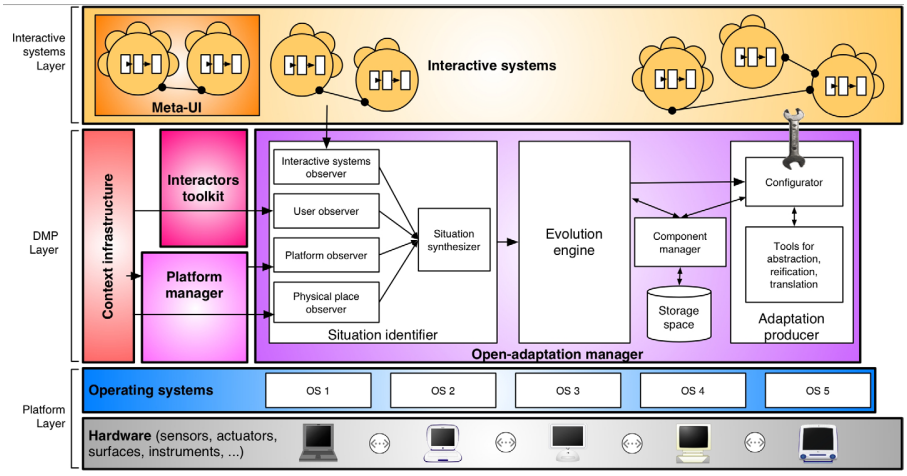

Figure 3.4: The Cameleon RT architecture reference model (Balme et al., 2004).

leased in ubiquitous computing environments. Their reference model covers user interface distribution, migration, and plasticity (adaptable to context and still usable).

As shown in Figure 3.4, the architecture defines three layers of abstraction. The middle layer provides context sensing and adaptation. The bottom layer consists of physical hardware as well as operating system. The interactive applications and a meta-user interface are contained in the top layer. This meta-user interface provides metadata about the user interface and allows a user to control the behavior of the middle layer. A flower-like shape of the figure denotes open-adaptive components. The miniature adaptation-manager shape denotes close-adaptive components. Arrows denote information flow, and lines bi-directional links.

DynaMo-AID DynaMo-AID (Dynamic Model-Based Used Interface Development) (Clerckx et al., 2004) is a design process and a runtime architecture to develop context-sensitive user interfaces that

support dynamic context changes. DynaMo-AID is part of the Dygimes (Coninx et al., 2003) User Interface Creation Framework.

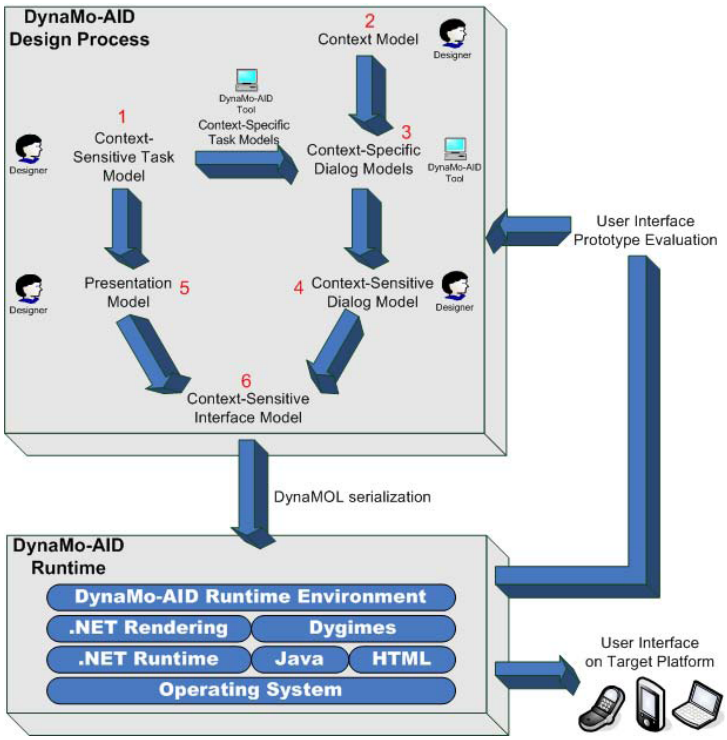

Figure 3.5: DynaMo-AID Development Process (Clerckx et al., 2005).

Figure 3.5 shows an overview of the DynaMo-AID development process. First designers have to specify declarative models describing the interaction. These models are: *context-sensitive task model* describing the tasks user and application may encounter, *context model* denoting what kind of context information can influence the interaction, *context-specific dialog models* that are automatically extracted from the task model for each context of use, *context-sensitive dialog model* that is the context-specific dialog model with extra edges describing possible context changes,

presentation model describing how the interaction should be presented to the user, and *context-sensitive interface model* that is the set of previously model components.

Next, the models can be serialized to an XML-based high level user interface description language in order to export the models to the runtime architecture. The designer can then render a prototype and adjust undesirable aspects that came to light in the prototype. After this iteration, the final user interface can be deployed on the target platform.

FAME. FAME (Duarte & Carriço, 2006) is a model-based Framework for Adaptive Multimodal Environments. The framework expands on previous frameworks and models, capturing the process of adaptive multimodal interface analysis. It is not intended to be a tool for automatic application development.

The architecture proposed by FAME (see Fig. 3.6) uses a set of models to describe relevant attributes and behaviors regarding user, platform and environment. The information stored in these models, combined with user inputs and application state changes, is used to adapt the input and output capabilities of the interface. To assist in the adaptation rules development, the concept of behavioral matrix is introduced. The matrix reflects the behavioral dimensions in which a user can interact with an adaptable component.

Two levels are identified in FAME's architecture. The inner level, or adaptation module, comprised of the different models and the adaptation engine, is responsible for updating the models and generating the system actions. The outer level, corresponding to the multimodal application layer, is responsible for the multimodal fusion of user inputs, transmitting the application specific generated events to the adaptation core, executing the multimodal fission of the system actions, and determining the presentation's layout.

The adaptation is based in three different classes of inputs: user

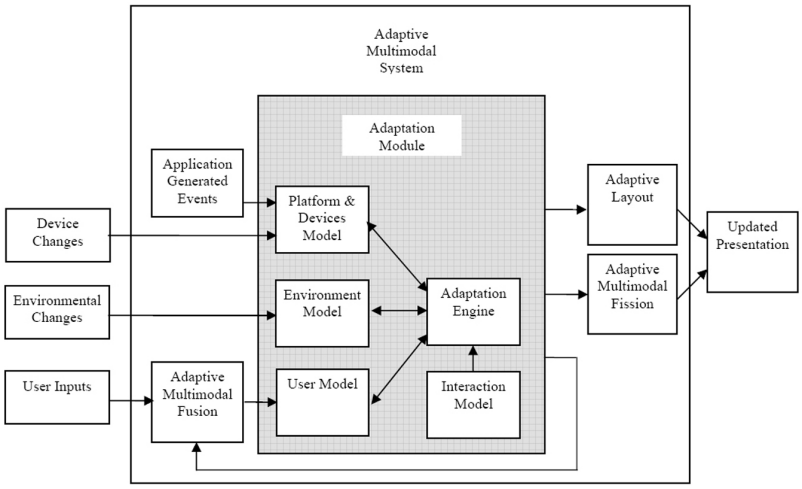

Figure 3.6: FAME's architecture (Duarte & Carriço, 2006).

actions, application generated events and device changes, and environmental changes.

3.1.1 Analysis and discussion

The different proposals described follow a model-based approach for describing interaction and developing context-aware user interfaces for mobile computing. These approaches are focused on describing abstract aspects of interaction as user tasks or the dialog between the user and the system. Thanks to the use of MDE techniques (Schmidt, 2006) some of the steps in the development of user interfaces can be automated making easier prototyping. However, in practice, the resulting interface usually requires different changes at implementation level in order to fit customer expectations (Pederiva et al., 2007). If the user interfaces are modified at code level it is dificult to keep the consistency for aspects such as adaptation that impact on many different elements in the system. Considering the need to manipulate user interfaces at a concrete level, we think that is is important to provide mechanisms

for representing the interface at an abstraction level that represents the result obtained but still hides the complexities of the underlying technology. The definition of a model that is close to the technology has been considered by different approaches. Model Driven Architecture (Miller & Mukerji, 2003) defines a Platform Specific Model (PSM) as a view of a system from the platform specific viewpoint.

Although these proposals presented recognize the need for a concrete user interface model, efforts have been put on representing structural aspects of the user interface, overlooking adaptation aspects that we have taken into account. We propose to represent behaviour aspects in the design interface level in order to represent how the different elements are adapted when a particular situation is produced.

Research prototypes to support model-based user interface processes resulted in tools where models can be constructed and mappings can be calculated. However the use of graphical notations, giving the designer an intuitive overview of the models is still limited (Van den Bergh & Coninx, 2004). Most of the time not all the models are visualized equally thoroughly or not visualized at all, and connections between distinct models are left behind in design tools. Our approach provides graphical tools for all the steps, guiding the designer in the development process.

These proposals consider the context of use as three dimensions: the user, the platform and the environment. Our proposal introduces user attention in the adaptation process as well as the three dimensions mentioned. So, services can provide their functionality at different obtrusiveness levels according to the context of use.

3.2 Non-intrusive mobile computing

Mobile computing devices will increasingly deliver phone calls, reminders, email, task lists, instant messages, news, and other time and/or place-based information. These are contributing to feelings of information overload and unexpected interruptions. Moreover, new sensor-enabled mobile devices will permit applications that proactively deliver information to people when and where they need it. Proactive promptings

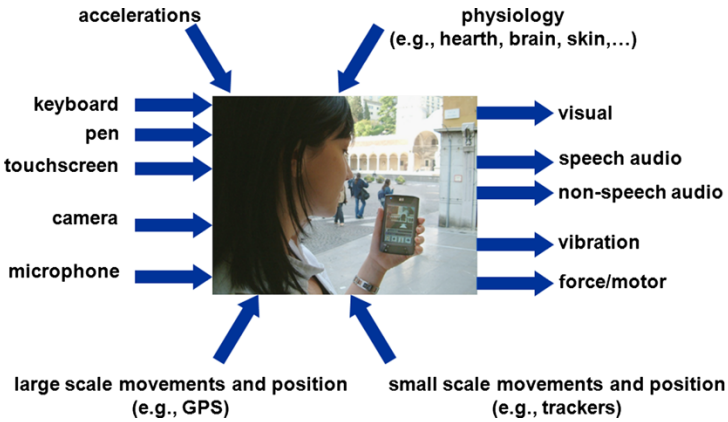

Figure 3.7: Input and output channels in mobile multimodal interfaces (Chittaro, 2010).

could contribute to feelings of interruption and annoyance.

Each time a device proactively provides information, it is competing for the user's attention and possibly interrupting the ongoing tasks. Research in this area are focused in determining a good time to interrupting the user.

Figure 3.7 summarizes the many channels that can be currently exploited to send and receive information from a mobile device. Choosing the appropriate modality or combination of modalities can help in reducing the attentional demand. The approaches studied from this area are detailed below.

Toward more sensitive mobile phone. Hinckley and Horvitz (Hinckley & Horvitz, 2001) described sensing techniques intended to help make mobile phones more polite and less distracting. They stated that many interactions with cell phones can be demanding of cognitive and visual attention. They modeled interruptibility by considering the user's likelihood of response and the previous and current activity.

Three sensors necessary to detect these factors were built in a mobile computing device: a two-axis linear accelerometer for tilt

sensing, a capacitive touch sensor to detect if the user was holding the device, and an infrared proximity sensor that detected if the head was in close proximity to the device.

The experiments performed include choosing an appropiate notification modality for an incoming call and reducing attentional demands to answer the calls.

SenSay: a context-aware mobile phone. Sensay (*sen*sing & *say*ing) (Siewiorek et al., 2003) is a context-aware mobile phone that modifies its behavior based on its user's state and surroundings. It adapts to dynamically changing environmental and physiological states and also provides the remote caller information on the current context of the phone user.

To provide context information SenSay uses light, motion, and microphone sensors. The sensors are placed on various parts of the human body with a central hub, called the sensor box, mounted on the waist. SenSay introduces the following four states: *uninterruptible*, *idle*, *active*, and the default state, *normal*. A number of phone actions are associated with each state. For example, in the uninterruptible state, the ringer is turned off.

SenSay can either eliminate unwanted interruptions or actively notify the user of an incoming call by adjusting ringer and vibrate settings. It also has te ability to relay the user's contextual information to the caller when the user is unavailable and it leverages idle time periods by making call suggestions to the user based on call history. The resuls of the experiment showed that clear delineations can be made between different user states from the data.

Interruptions during activity transitions. In this study (Ho & Intille, 2005), a context aware mobile computing device was developed that automatically detects postural and ambulatory activity transitions in real time using wireless accelerometers. This device was used to experimentally measure the receptivity to interruptions delivered at activity transitions relative to those delivered

at random times.

The hypothesis was the possibility that prompts from mobile devices may be perceived as less disruptive if they are presented at times when the user is transitioning between different physical activities. The experiment was motivated by the casual observation that a transition between two different physical activities may strongly correlate with a task switch, and a task switch may be a better time to prompt the user with an interruption than an otherwise random time.

In the study, notifications that were delivered during activity transitions were generally found to be more easily accepted by the participants. In the case of everyday activities, user engagement in activities is expected to be lower when transitioning between activities, resulting in a high acceptability of notifications. Thereby suggesting a viable strategy for context-aware message delivery in sensor-enabled mobile computing devices.

3.2.1 Analysis and discussion

In Section 2.3 of the previous chapter, we introduced several initiatives that have studied strategies to minimize the burden of interruptions. These approaches are extended in this chapter, focusing on those that are carried out in mobile devices. All of these initiatives have studied strategies to minimize the burden on interruptions in mobile computing. However, these initiatives are almost exclusively focused on evaluating the adequate timing for interruptions, while user interface adaptation or presentation mode has received few attention.

In this work, we focus on choosing the best interaction mechanism for a given context of use. The techniques presented can be used in conjunction with our approach to choose the better time for interrupting the user.

3.3 Attentive User Interfaces

Computing interfaces that are sensitive to the user's attention are called Attentive User Interfaces (AUIs) (Vertegaal, 2003). Attentive User Interfaces, by generating only the relevant information, can in particular be used to display information in a way that increase the effectiveness of the interaction (Huberman & Wu, 2007).

Roel Vertegaal of Queen's University has been pioneering the design of user interfaces that are attentive to their user. These *attentive user interfaces* optimize the allocation of the attentive resources of users and systems; they enable devices that do not bug you when you are busy. Cues of user attention are applied to make devices more sociable and efficient and to increase the bandwidth of user communication with and via computers.

AUIs use specific input, output and turn-taking techniques to determine what task, device or person a user is attending to. This is done by detecting a user's presence, orientation, speech activity and gaze and statistically modelling attention and interaction in order to establish the relevance of information that could be presented to the user and the urgency of doing so in the context of the current estimated activity. The research in this area is detailed below.

A Framework for Attentive User Interfaces. Vertegaal et al. (Vertegaal et al., 2006) presented a framework for augmenting user attention through attentive user interfaces. They extended the GUI elements to interactions with ubiquitous remote devices, drawing parallels with the role of attention in human turn taking. This is shown in Fig 3.8. Windows and icons are supplanted by graceful increases and decreases of information resolution between devices in the foreground and background of user attention; Devices sense whether they are in the focus of user attention by observing presence and eye contact; Menus and alerts are replaced by a negotiated turn taking process between users and devices. Such characteristics and behaviours define an attentive user interface.

The framework are based on five key properties of AUIs: (1) *Sens-*

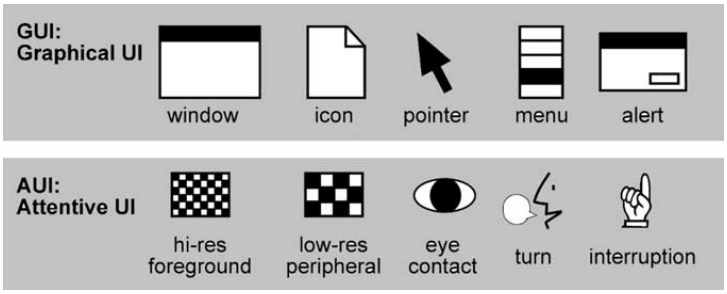

Figure 3.8: Equivalents of GUI elements in Attentive UI (Vertegaal et al., 2006).

ing attention by tracking user's physical proximity, body orientation and eye fixations, (2) *Reasoning about attention* by statistically modeling simple interactive behavior of users, (3) *Communication of attention* to other people and devices, (4) *Gradual negotiation of turns* to determine the availability of the user for interruption, and (5) *Augmentation of focus* with the goal of augment the attention of their users.

Models of attention in computing and communication. Horvitz et al. (Horvitz et al., 2003) review principles for sensing and reasoning about a user's attention with Bayesian models, and for using these inferences to identify ideal decisions in messaging, interaction, and computation.

On the one hand, in order to access and use information about a user's attention, they constructed by hand and learned from data Bayesian models viewed as performing the task of an automated "attentional Sherlock Holmes", working to reveal current or future attention *under uncertainty* from an ongoing stream of clues. Bayesian attentional models take as inputs sensors that provide streams of evidence about attention and provide a means for computing probability distributions over a user's attention and intentions.

On the other hand, in order that computers and communications

systems have awareness of the value and costs of relaying messages, alerts, and calls to users, they have presented the *Notification Platform*. The Notification Platform system modulates the flow of messages from multiple sources to devices by performing ongoing decision analyses. These analyses balance the expected value of information with the attention-sensitive costs of disruption. The system employs a probabilistic model of attention and executes ongoing decision analyses about ideal alerting, fidelity, and routing.

Personal attentive user interfaces. Streefkerk et al. (Jan Willem Streefkerk & Neerincx, 2006) proposed to design personal attentive user interfaces (PAUI) for which the content and style of information presentation was based on models of relevant cognitive, task, context and user aspects. They presented a user-centered design (UCD) method for the iterative development and validation of the proposed PAUI. In UCD, reasoning about prospected use of a system gives valuable insights on and validity to user requirements and collaboration styles.

The relevant cognitive aspects considered are attention, working memory, task switching, and situation awareness. Regarding to task aspects, they analyzed information access, prioritization, and notification. Context aspects taken into account were location, time and environment. Finally, the relevant user aspects were preferences, duties, expertise and individual differences.

The approach presented to designing for attention was based on the usability engineering approach which incorporates scenario-based design (Carroll, 2000). The process starts with the definition of a concept, which is a general description of the proposed system. Then, a scenario is drafted from the relevant usage context. From this scenario collaboration styles are defined which indicate how the system interacts with the user. Next, user requirements are derived. These requirements will be based on the relevant cognitive, task, and context aspects. Finally, collaboration styles and user requirements form the basis of the features.

Assesment of the collaboration styles, user requirements and features is done by validating them to objective quality criteria, such as established HCI metrics.

3.3.1 Analysis and discussion

The different approaches in the attentive user interfaces area are mainly focused on detecting or inferring attention. One of the problems we encountered is that the sensing technologies used for sensing attention may be invasive. Another issue in which users can feel unreliability is the prioritization of notifications, since automated services rank and prioritize the information they receive. Also, user may want to understand why certain adaptations were made, especially when these adaptations are not consistent over time, place and user. But, as mentioned in (Vertegaal et al., 2006), the most pressing issue relating to the sensing technologies of attentive user interfaces is that of privacy. How do we safeguard privacy of the user when devices routinely sense, store and relay information about their identity, location, activities, and communications with other people?

The presented approaches do not provide neither modeling languages nor tools for the development of this kind of user interfaces. They lack methodological guidance and do not provide tool support for the easy user interface definition. Our approach provides mechanism for the design and development of user interfaces in a declarative manner and different visualizations of the adaptation process.

3.4 Conclusions

This chapter presents the state of the art in the disciplines that are related to this work. These areas are really active these days with many initiatives. However, there is a lack of proposals to provide mechanisms that allow the development of user interfaces within the three aplication domains. Modeling languages are used only to describe context-aware user interfaces which do not take into account user attention. Towards

creating systems that adapt their level of intrusiveness to the context of use, researchers and designers face a design challenge in terms of creating devices that sense user's state and attention in order to calculate the better time for interruptions.

The present work provides a modeling language to support the design and development of mobile services that can be adapted to the attentional demand. With the modeling language we can describe by means of graphical notation the user interface elements that are involved in the adaptation process, and the circumstances under which they must be adapted. Since many approaches address the problem of detecting or inferring attention as well as user's state, our approach is not focused on capturing this information but describing the way interaction is adapted according to this.

Chapter 4

Designing unobtrusive mobile interactions

The increasing of a wide variety of powerful mobile devices and the heterogeneity among them introduces new challenges in the design and development of mobile services. All these services compete for the attentional resources of the user. Thus, it is essential to consider the degree in which each service intrudes the user mind (i.e., the obtrusiveness level) when services are designed.

Mobile interaction design introduces additional challenges compared to the design of traditional desktop applications. Most of the new difficulties for mobile interaction design are due to (1) the diversity of usage contexts and their constant changes during user activities, and (2) the specific characteristics that define mobile devices such as small screen, lack of keyboard, etc (de Sá & Carriço, 2009). For the design of adaptation, designers must define *what* changes in the user interface and *why* these changes are produced.

This chapter introduces a methodological approach for the design of adaptation aspects for mobile interaction. The goal of this method is to

provide a mechanism **for defining the desired degree of automation for the interaction of a given mobile service**. In order to systematize the development of such services and achieve a well-designed system, the method is based on the user-centered design principles.

User Centered Design (UCD) (Mao et al., 2001) is a design approach that focuses on the needs of the end user. It is a process in which the needs, wants, and limitations of end users of a product are given extensive attention at each stage of the design process. In UCD, reasoning about prospected use of a system gives valuable insights on and validity to user requirements and interaction styles. Using a scenario from the application domain, interaction styles and user requirements are formulated, based on the relevant aspects. *Design* begins with mockups and storyboards, and progresses to interactive *prototypes*. *User feedback* is gathered at every stage of the process, and the design is iterated on until it meets the requirements of the end user. Because design changes are applied to a prototype, iteration occurs very rapidly. The result is a prototype that serves as an interactive specification for the product.

A User Centered Design paradigm is followed by ubiquitous computing systems since computing resources become invisible to the user providing a *natural interaction* with the system. For this reason, user has to drive the design, helping to take decisions and validating the design.

Figure 4.1 shows an overview of the stages in the development process proposed in our approach. Mobile services are iteratively designed. This means that design and development is iterative, with cycles of "design, test, measure, and redesign" being repeated as often as necessary. In each design cycle, the adaptation of the services is put into practice to obtain feedback from the users. The feedback obtained is used to improve the original design. When no further changes are required, the system specification is used to guide the implementation of the final system. This chapter provides detail on the stages involved in the iterative design of context-aware mobile services. It introduces the aspects to be considered during design. Chapter 5 provides detail on the prototyping to evaluate the design and the implementation of the unobtrusive mobile services.

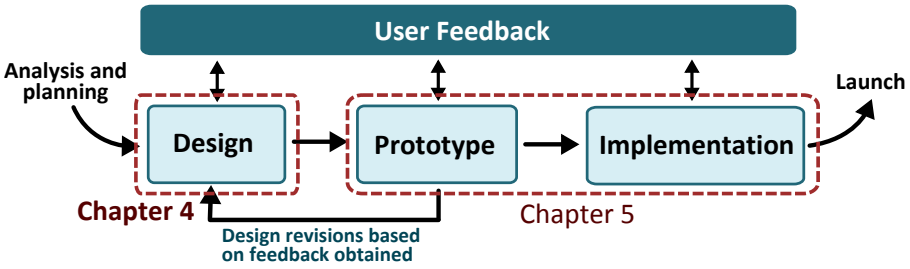

Figure 4.1: The stages proposed in the user-centered development process.

The remainder of the chapter is structured in the following manner. Section 4.1 provides an overview of the development method. Section 4.2 introduces the concepts used in the design stage to describe the adaptation in an abstract manner. Section 4.3 provides tool support for the definition of the system specification that is used as input for later steps in the development process. Section 4.4 concludes the chapter.

4.1 Development method overview

This section provides a more detailed description on the development method introduced in this work. The design stage is the initial stage in our method (see Fig. 4.1). Since we are following a model-driven approach, the specification obtained at design drives the later stages in the development of the system. Thus, the design becomes central to the development method.

The design method captures by means of models the concepts that are relevant in the development of context-aware interaction. The benefits of using a model-driven approach and the steps followed in the development of unobtrusive mobile interaction are introduced below.

4.1.1 Why a modeling approach?

Human-Computer Interaction (HCI) community has considered the use of models to describe interaction for long. In a context where the possible combinations of users, situations and devices are constantly increasing, the implementation of ad-hoc solutions to cover all possible combinations is not feasible. Sottet (Sottet et al., 2005) reports this problem and stresses the relevance of Model Driven Engineering for the modeling of interaction in Ubicomp systems. MDE proposes the use of models to specify the desired aspects of a system, and then, derives the actual code in an automatic way. The specified system can be automatically generated for different platforms from an abstract description.

Abstraction is one of the fundamental principles of software engineering in order to master complexity (Kramer, 2007). Our approach makes use of modeling techniques in order to promote abstraction in the context-aware user interface development. By abstracting technical details, we can describe interaction components and contextual information regardless of the particular technology used for the implementation. In the case of the development of user interfaces, modeling techniques are applied to obtain the following benefits (Silva, 2000):

- Models can provide a more abstract description of the UI than UI descriptions provided by the other UI development tools.
- Models facilitate the creation of methods to design and implement the UI in a systematic way since they offer capabilities: (1) to model user interfaces using different levels of abstraction; (2) to incrementally refine the models; and (3) to re-use UI specifications.
- Models provide the infrastructure required to automate tasks related to the UI design and implementation processes.

Benefits of our method

The present modeling solutions for interaction usually describe **which** information is presented to the user by means of an *Abstract User In-*

terface, and then define a discrete set of platforms, environments and user types to determine **how** the interaction will be offered for each set of context conditions (Calvary et al., 2003). But this discretization of context conditions presents some problems:

- **Similarities between the different context conditions are not exploited.** Context conditions are considered to be atomic without taking into account the existence of shared limitations and capabilities. For example, a mobile phone and a PDA both have a limited screen, so the interaction with the system would be more similar in these devices compared to the interaction offered in a desktop computer or a device with no screen at all (see on the left of Fig. 4.2).

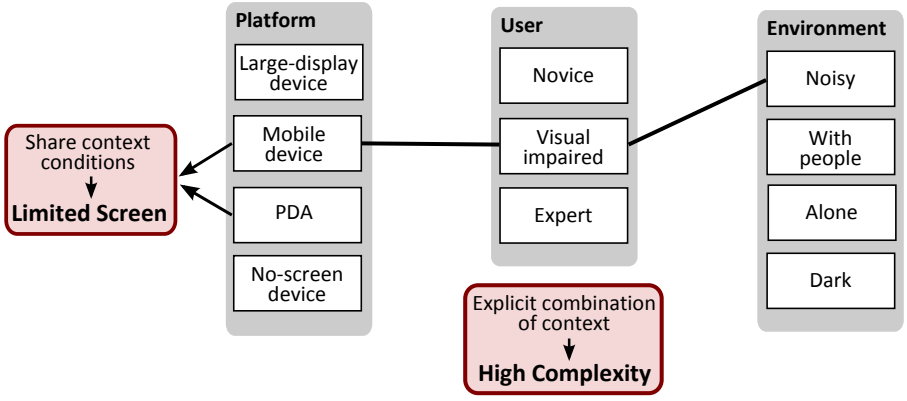

Figure 4.2: Problems of context condition discretization.

- **All combinations of context conditions are considered explicitly to define the interaction.** This implies specifying how interaction is derived from an Abstract User Interface for each platform-user-environment combination. For example, we must consider how to produce the interface for a visually-impaired user accessing the system from a mobile device platform in a noisy

environment. Therefore, the complexity of the definition of interaction increases with the number of context conditions considered (see Fig. 4.2).

Since the promise of natural interaction of Ubicomp implies adapting to a large number of context conditions, we propose decomposing the context conditions in their *features* -capabilities and limitations-, and using these features to describe the interaction in an abstract manner. In this way, the above problems are solved. The features can be shared among context conditions to indicate their commonalities and differences. Figure 4.3 shows an example of the proposal. Context X has the features *F1, F2, F3* and *F4*, and the last three features (*F2, F3, F4*) are common to another context Y. We aim to express the interaction as a function of features. In this way, we could support contexts X and Y, and all another contexts that could be expressed as a combination of these shared features.

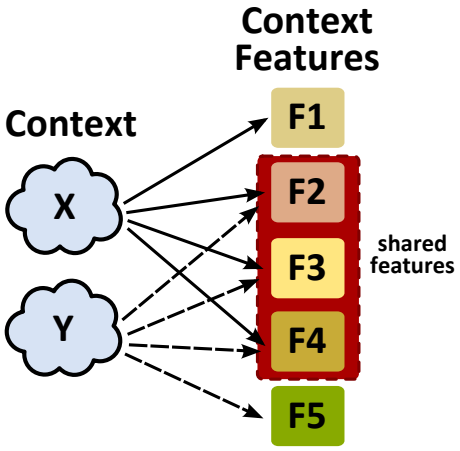

Figure 4.3: Context decomposition into features.

For example, a noisy context and a user with an auditory impairment require interaction not to be provided by means of audio. By considering the specification in terms of features (as opposed to specifying for each context), the duplication of efforts in the development are minimized

since both cases are expressed as the exclusion of the auditory feature. An overview of the steps involved in the development process is provided below.

4.1.2 Proposal overview

The development method proposed in this work supports the (1) design, (2) prototyping and (3) implementation of unobtrusive mobile services. The method defined involves three development roles: *Analysts*, *Designers* and *Developers*. Each one makes decisions at different abstraction levels. The steps performed in each phase of the cycle are detailed below:

1. **Design.** In the design stage, the aspects that are considered relevant for the final system are captured and they are used to guide the implementation. In particular, the goals of this stage for the development of unobtrusive mobile services are to (1) detect the user needs to determine the obtrusiveness level required for the interaction and (2) make use of the adequate interaction mechanisms to provide the functionality according to the obtrusiveness level. This information is captured in models. The models obtained at this point are the input to the system implementation. Further detail on the concepts captured in this stage is provided in Section 4.2

2. **Rapid prototyping and evaluation.** Once designers consider that the behavior defined for adaptation is the one desired, they can use the models defined to elaborate a prototype. This prototype can be used to evaluate the adaptation of the system to one or several simultaneous factors. The obtained interface can be automatically calculated from the models defined regardless of the number of simultaneous factors considered. Feedback is gathered from end-users in order to determine whether the proposed mobile services improves the existing in terms of system usability and interaction adaptability. Chapter 5 provides the guidelines to perform the evaluation of the prototype.

3. **Implementation.** The models obtained at this point are the input to the next stages of the UCD cycle. Once the models have been adjusted to fit with the user needs, a final software solution can be obtained. The derivation of a software solution from the system specification is described in Chapter 5.

By capturing user needs and adaptation requirements for the user interface in different models, model-based tools can be used to define, validate and guiding the development of these unobtrusive adaptable user interfaces. Once the models have been adjusted to fit with the user needs, a final software solution can be obtained.

In the following section we provide further detail on the design stage, describing the concepts that are captured to specify the unobtrusive mobile services. These concepts are later used to obtain an implementation.

4.2 Design stage

In a mobile context where users are permanently connected to the environment, users may be interrupted often. Services should interact with users in a way that is not disturbing for them. So, the goal of the approach presented in this work is to manage the attentional demand of services according to each user and the environment in order to provide an unobtrusive interaction. In order to avoid service behavior from becoming overwhelming, we propose a technique to adjust the way attentional resources of each user are used by pervasive services.

In order to determine the adequate interaction technique to be used in each service, the automation level required for the service and the available interaction mechanisms are studied in an abstract manner. Each service in the system is analyzed to determine to which extend the interactions involved must intrude the user mind (the obtrusiveness level). In this way, the degree of automation of the services can be adjusted to the particular requirements of each user and the appropiate interaction technique can be selected from the ones available. Some

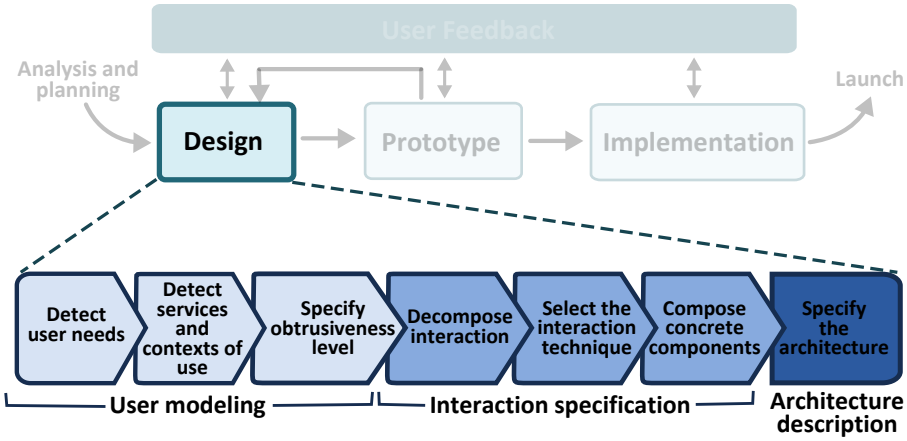

Figure 4.4: The different tasks in the design method proposed.

tasks could require implicit interaction in order to allow the process to be more fluent, while others would require the users to be completely aware of the system actions.

This section introduces the modeling techniques applied for describing this unobtrusive mobile interaction by adjusting the obtrusiveness level required for each service. Figure 4.4 details the tasks involved during the design stage. The steps performed in this design phase are detailed below:

1. **User modeling.** First, *analysts* are in charge of understand and specify **who** the users will be by studying their cognitive, behavioral and attitudinal characteristics. Understanding behavior highlights priorities, preferences, and implicit intentions (Sharp et al., 2007). To understand the users and capture their needs and preferences we use **personas** (also known as user profiles). We have used personas since it provides features that directly address specific user needs and it is an habitual technique used in the design of user centered approaches. From the definition of personas, analysts have to **detect the services** of the system to

achieve user needs. Services become the basic unit of work during development. Then, the **contexts of use are studied** and services are designed to support it and the user needs.

Once user needs, services and contexts of use are identified, *designers* must define the way in which a pervasive service is accessed by **adjusting its obtrusiveness level** according to the user needs and contexts of use.

2. **Interaction specification.** Then, the commonalities and differences between adaptation aspects are specified by means of **interaction features**. By considering the desired obtrusiveness level, an adequate **interaction mechanism is selected** to provide the functionality according to the obtrusiveness level. Note that depending on the persona and the current activity, a different obtrusiveness level could be required for the same service. For example, the obtrusiveness level for the notification of a supermarket nearby can be changed depending on the user's activity, but can be also changed depending on the priority it has for the user (e.g., demanding more attention when the user is walking around the supermarket or when the number of items to buy exceed a fixed number).

   Once interaction requirements are specified in an abstract manner, *designers* have to describe the concrete interaction components that are going to represent the user interface elements available for a specific platform. **Composing these concrete components**, interaction features are going to be supported for a specific platform.

3. **Architecture description.** Finally, *designers* have to define the way **user interfaces are integrated with the different components** from the current and external systems. This allows to express the dependencies of an application in terms of data and functionality and detect the sources of context information that can trigger user interface adaptation. In order to do so, the components of the specific mobile platform used for the application are captured in a model.

The following subsections provide more detail about these design concepts.

4.2.1 User modeling

The first step in the design phase is to understand the users and capture their needs and preferences. This information will determine the obtrusiveness level required for a service. The tasks carried out in this stage are highlighted in Figure 4.5.

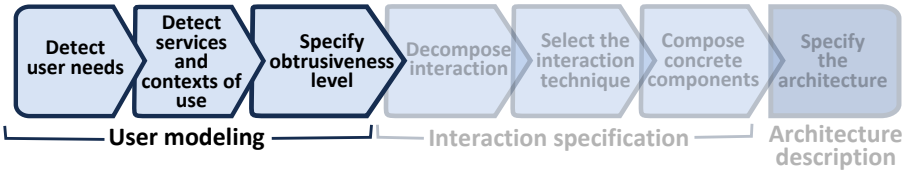

Figure 4.5: The different tasks in the user modeling stage.

Personas are used to gather the relevant information of the audience to help drive design and detect common functionalities between users. From the software engineering side, different mechanisms exist in order to define the relationship between users and their performed activities such as UML Use Case Diagrams (Rumbaugh et al., 1998), ConcurTaskTrees (Mori et al., 2002), and Business Process Modeling Notation (BPMN) (OMG, 2006). Personas are usually used in the design of user centered approaches. According to the users, personas give a much more concrete picture of typical users providing features that directly address specific user needs (Gulliksen et al., 2003). Thus, it is interesting the use of them in this work where we have directly address specific user needs.

Moreover, this work allows to express the user needs and contexts of use in terms of obtrusiveness. In order to describe the obtrusiveness level required for each service, we use the conceptual framework defined in (Ju & Leifer, 2008). In this way we can define to which extent

an interaction must intrude each user's mind and adapt the system accordingly.

1. Detecting user needs

The first step in the adaptation of pervasive services is to understand who the users will be. In order to do this we make use of **personas**.

Personas describe a target user of the system, giving a clear picture of how they are likely to use the system, and what they will expect from it (Brown, 2010). Personas have become a popular way for design teams to capture relevant information about customers that directly impact the design process: user goals, scenarios, tasks, and the like. Scenarios are little stories describing how typical user tasks are carried out. They help to anticipate and identify the decisions a user will have to make at each step in their experience and through each environment or system state they will encounter.

There is no standard format for personas, and different approaches are offered. Regardless of the approach selected, personas should express what users need and what they expect. In this work, we follow the notation defined in (Brown, 2010) to determine the needs of each user and the functionality required. Aditionally, we introduce the obtrusiveness concept expressing this functionality in terms of obtrusiveness.

In this notation, the information is structured following three layers of detail. Figure 4.6 shows the elements of a persona prioritized into this three layers. Layer 1 contains the fundamental elements to establish user requirements. These elements are: the *name* of the persona, some *key features* that distinguish the user group from others, a *descriptive dimensions* that are individual scales representing knowledge, tasks, interests and characteristics, the *objectives and motivations* of the persona within the scope of the system and annotations of the data *sources*. These elements can be complemented with information of the other layers such as the *concerns* of the personas that will influence their experience with the system, the *scenarios and circumstances* that set the stage for an interaction between a user and a system, the *personal background*, and a *photograph* of the persona.

Layer 1: Establishing Requirements	Layer 2: Elaborating Relationships	Layer 3: Making 'em Human
Name	Concerns	Personal Background
Key Distinguishing	Scenarios	Photo
Descriptive Dimensions	Quotes	System Features
Objectives & Motivations		Demographic Information
Source		Technology Comfort

Figure 4.6: The elements of a persona prioritized into three layers

According to these elements that characterize a persona, we can define the functionalities and tasks that user needs to achieve their objectives and motivations. Moreover, we can detect common functionalities between personas and express these functionalities in terms of obtrusiveness. Establishing the degree of user attention that a task need, we avoid to develop overwhelming services.

Figure 4.7 shows an example of a persona for a Smart Home system. This persona gives a detailed picture of a typical "busy user" that wants to use Smart Home services for optimize his time. This excerpt of a persona provides the basics of a user's needs and behaviors. From this persona, we deduce that the user has an advanced profile that wants a Smart Home services for helping him in home tasks to do not waste time, he wants be aware of pending tasks related to home and work and he hopes to be alerted of the updates that services perform. He prefers as many services as possible.

Some other personas could require other functionalities and different information presentation and interaction with the services depending on their needs. Thus, personas will guide subsequent adaptations in information presentation, modality and interaction style. By the definition of personas, we can detect functionalities and we can also determine different contexts of use for the services. In the following section, we

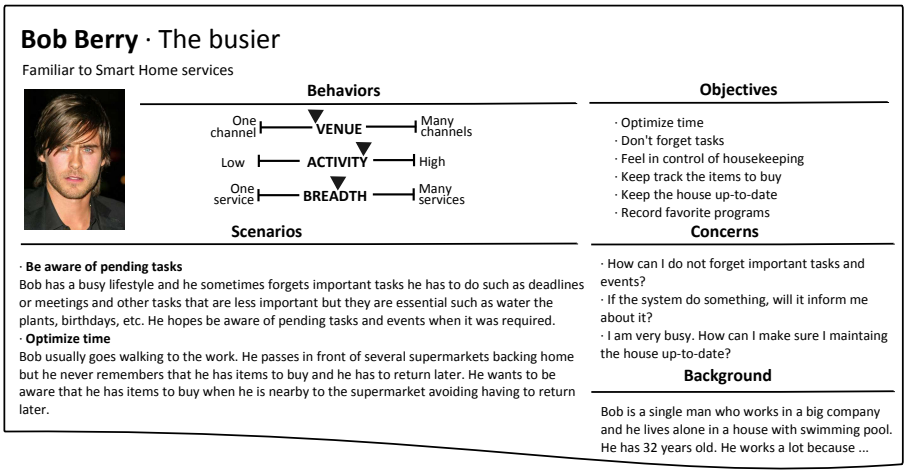

Figure 4.7: Excerpt of a persona

describe the different kinds of context that can be considered to adapt the services.

2. Detecting services and contexts of use

Once user needs and the functionalities are identified, we have to determine the services that are going to give support to these functionalities.

For the example of persona defined previously (see Figure 4.7), the services detected are: a shopping list to keep track the items to buy, an agenda that notifies him important tasks, a video recorder that records his favorite programs, and a supermarket notification to remember him that he has items to buy.

Aside from the user needs, mobile information systems are characterized by frequent changes in the context of the user. Different kinds of context can be considered when a mobile system is developed (Maiden, 2009): *computing context*, *environmental context*, *user context*, and *time context* (see Figure 4.8). The *computing context* is everything related to computational resources, such as available networks, network band-

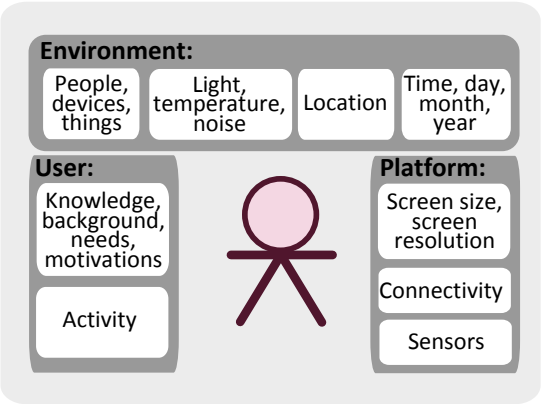

Figure 4.8: Sources of contextual information

width, communication costs, and nearby computational resources. The *environmental context* involves factors in the environment of the device with which the user interacts, such as temperatures, noise levels, and speed and lighting levels. It can include the *time context* information. *Time context* captures information such as absolute time, date, and day of the week. The *user context* is information about the user interacting with the device, such as a profile (for example, age, expertise, needs, motivations and preferences), location (for example, geographic position), and proximity (for example, distance to another person).

In this work, the adaptation of the obtrusiveness level can be the result of changes in these sources of contextual information. In particular, in this work we have taken into account contextual information such as user location, mobile location or the engagement of the user in other activities. According to this contextual information and the user needs, services will be adapted in terms of obtrusiveness. For example, if the user is engaged in an important activity, a notification will appear in a subtle manner avoiding to disturb him/her.

In order to adapt the services and provide them in an unobtrusive manner for each persona, we define the services in terms of obtrusiveness. In the following section we provide a detailed description of the

obtrusiveness concept as it has been defined for the present work.

3. Specifying the obtrusiveness level

The adaptation space is normally defined in terms of some relevant properties for the system. For example, some user interface adaptation approaches define their adaptation space in terms of the *environment* and the *platform* properties (Calvary et al., 2003). When this criteria is followed, different variants of the system are provided to the user depending on whether a desktop or a mobile device (platform) is used; of the system is used in a noisy or a quiet room (environment). Our approach defines the adaptation space for the system in terms of *obtrusiveness*.

With more devices added to our surroundings, users increasingly seek simplicity (Maeda, 2006). Since user attention is a valuable but limited resource, an environment full of embedded services must behave in a considerate manner, demanding user attention only when it is required (Gibbs, 2004). In the mobile environment, interaction must be adapted in order not to intrudes on user attention. It is the task of the designer to determine which tasks the system can perform automatically and which ones must the user be aware of.

We make use of the conceptual framework presented in (Ju & Leifer, 2008) to determine the **obtrusiveness level** for each interaction in the system. This framework defines two dimensions (see Fig. 4.9) to characterize implicit interactions: *initiative* and *attention*. According to the *initiative* factor, interaction can be *reactive* (the user initiates the interaction) or *proactive* (the system takes the initiative). With regard to the attention factor, an interaction can take place at the *foreground* (the user is fully conscious of the interaction) or at the *background* of user attention (the user is unaware of the interaction with the system).

Other frameworks exist for the definition of implicit interactions (Buxton, 1995; Horvitz et al., 2003). However, we found it very useful to consider *initiative* and *attention* as independent concepts. In the case of mobile interaction adaptation, automation and user awareness are factors that usually vary independently from service to service.

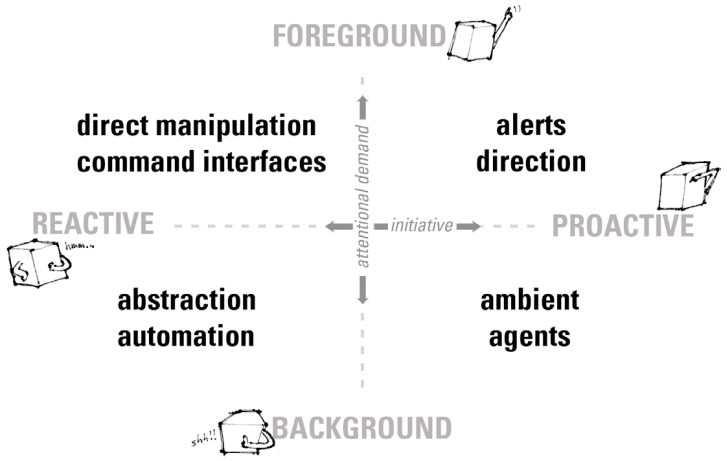

Figure 4.9: Framework for characterizing implicit interactions (Ju & Leifer, 2008).

According to our proposal, once services of the pervasive system are defined based on personas, designers must indicate the obtrusiveness level for each service depending on the user needs and the contexts of use. For the application of our proposal, we introduce an order in the values that define the *initiative* and *attention* axes. On the one hand, the extreme values for the *attention* axis are *Background* and *Foreground*. Since this axis represents user attention demands, we could order these values as $Background < Foreground$ to indicate that *Foreground* interactions require more attention than *Background* interactions. On the other hand, the *initiative* axis is related to automation, so we consider that the *Reactive* value provides lower degree of automation than the *Proactive* value (i.e., $Reactive < Proactive$).

A consequence of introducing this ordering in our approach is that we can express changes in the obtrusiveness level as increments and decrements in the different axes. The only rule that must be followed when dividing an axis is that the ordering must be preserved in each axis for the defined values.

Figure 4.10 illustrates an example of different services in the obtru-

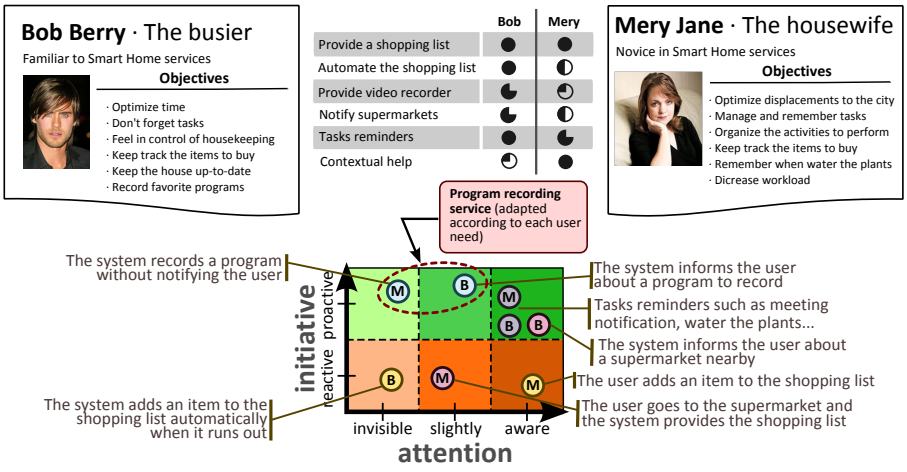

Figure 4.10: Services at different obtrusiveness level

siveness space for two personas. The different services were detected from personas. In order to highlight the similarities and differences of the needs and tasks of each persona we can create a simple table of needs comparison, using circles with shaded pie pieces to indicate the priorities. This table indicates the relative level or importance of a task for each persona.

In this particular example, the initiative axis is divided in two parts: *Reactive* and *proactive*. The attention axis is divided in three segments which are associated with the following values: *Invisible* (there is no way for the user to perceive the interaction), *slightly-appreciable* (usually the user would not perceive it unless he/she makes some effort), and *user-awareness* (the user becomes aware of the interaction even if he/she is performing other tasks). Designers can divide the obtrusiveness space into many disjoint fragments as they need to provide specific semantics.

In the example, we can see that the same activity for different personas makes sense to be in different obtrusiveness level because their needs are different. For example, for Bob the system is more likely to add an item to the shopping list automatically because he prefers

to automate the shopping list. However, the same service for Mery is completely aware since she prefers to add the items manually. Another example is the service to record programs. For Bob, this task is really important because he does not have time to see his favorite programs and he prefers the system records automatically the programs. For Mery, this is not very important because she has time to see the programs she likes. She prefers that the system informs her about to record a program. Although the general relevance of a service can be the same for different users at design time, the relevance varies on the different executions of the services. For example, we have considered the notification service to be relevant for Bob and Mery, however, they are not equally prone to be interrupted by the same kind of notifications (e.g., watering the plants or meeting notifications).

Nevertheless, these preferences can also change from time to time due to changes in the context. For example, the obtrusiveness level for the notification of a supermarket nearby can be changed depending on the user's location, but can be also changed depending on the priority it has for the user (e.g., demanding more attention when the supermarket is closer or when items to buy exceed a fixed number).

4.2.2 Interaction specification

The following step is to make use of the adequate interaction mechanisms to provide the functionality according to the obtrusiveness level. The tasks carried out in this stage are highlighted in Figure 4.11

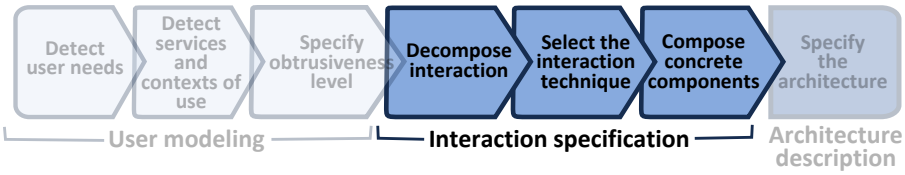

Figure 4.11: The different tasks in the interaction specification stage.

On the one hand, for the selection of the adequate interaction mech-

anism we make use of Feature Models. Feature Models allow to describe the essential aspects of each technology and the ways in which they can be combined. In this way, the complexity introduced by the technology heterogeneity can be mastered. By providing an intensional description of the interaction possibilities (as opposed to an extensional description of all the possibilities), we avoid having to define the interaction for each combination of context, obtaining "common aspects" between context factors.

On the other hand, the concrete interaction components that are going to represent the user interface elements available for a specific platform have to be described. The composition of these concrete components is going to represent the final user interface. The following subsections provide more detail on these aspects.

4. Decomposing the interaction

To make use of the interaction mechanism that supports the adequate obtrusiveness level, this work proposes decomposing the context conditions and user needs (adaptation aspects) in their *features* (capabilities and limitations), and using these features to describe the interaction in an abstract manner.

The Feature Model is introduced to reflect the terms in which the interaction is perceived. With this model, analysts can capture functional and non-functional aspects that are relevant to the system interaction.

Feature Modeling is a technique to specify the variants of a system in terms of features (coarse-grained system functionality) (Czarnecki & Kim, 2005). The relevant aspects of each platform and the possibilities for their combinations are captured by means of the feature model. Features are hierarchically linked in a tree-like structure through variability relationships. There are four relationships related to variability concepts in which we are focusing:

Optional. A feature can be selected or not whenever its parent feature is selected. Graphically it is represented with a small white circle on top of the feature.

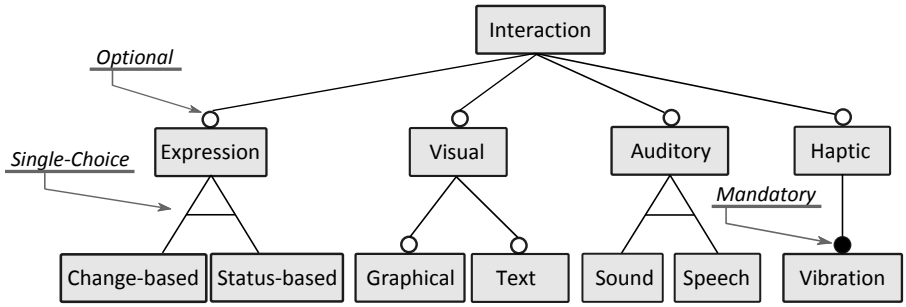

Figure 4.12: Interaction mechanisms Feature Model

Mandatory. A feature must be selected whenever its parent feature is selected. It is represented with a small black circle on top of the feature.

Or-relationship. A set of child features have an or-relationship with their parent feature when one or more child features can be selected simultaneously. Graphically it is represented with a black triangle.

Alternative. A set of child features have an alternative relationship with their parent feature when only one feature can be selected simultaneously. Graphically it is represented with a white triangle.

Besides describing the relevant aspects to the system, feature models have proven to be effective in hiding much of the complexity in the definition of the adaptation space (Cetina et al., 2009). Adaptation requirements can be described in a declarative manner instead of defining each particular combination. We make use of Feature Models to describe the possible interaction mechanisms and the constraints that exist for their selection. For example, according to our model showed in Fig. 4.12 an *auditory* element must either be *speech* or *sound*. In the same way, information or feedback can either be expressed *change-based*

(it reports only the changes) or *status-based* (it is continually informing about the status).

The definitions that are contained in the feature model are by no means considered universal. The Feature Model is intended to capture the perspective that analysts have about interaction. Features are only agreed abstractions that will be used to define the rest of the interaction models from a specific perspective. We can define the *visual* concept as "something that is perceivable using the eyes". In the example, we have considered that an interaction element is either visual or auditory, which is obviuosly a simplification since many common widgets normally combine these aspects (e.g., to offer feedback to the user).

In order to have a more precise notion of the role that the Feature Model plays in the specification of interaction, the consequences derived from the introduction of this model are detailed below.

Benefits of the decomposition into features

This work proposes the description of adaptation aspects by decomposing them into features. Discretizing the existing spectra of platforms is not an easy task. Mobile phones can be considered as a single platform with similar characteristics; however, this includes different kinds of devices: a conventional mobile phone, a phone with a GPS receiver, a phone with accelorometers and multi-touch screens, etc.

We propose to define adaptation aspects such as platforms, user needs or environments by the features that they support. As illustrated in Figure 4.13, a mobile device platform can be defined as a platform where the *auditory*, *visual* and *compact* is required. The same is valid for users, environments or any other kind of context conditions. In the example, the "noisy environment" is a kind of environment where the *auditory* interaction is not allowed. Feature models allow us to decompose the interaction in different aspects without explicitly having to define adaptation for each possible combination of context conditions. This avoid duplicating efforts in the development.

In addition, expressing adaptation aspects by means of features en-

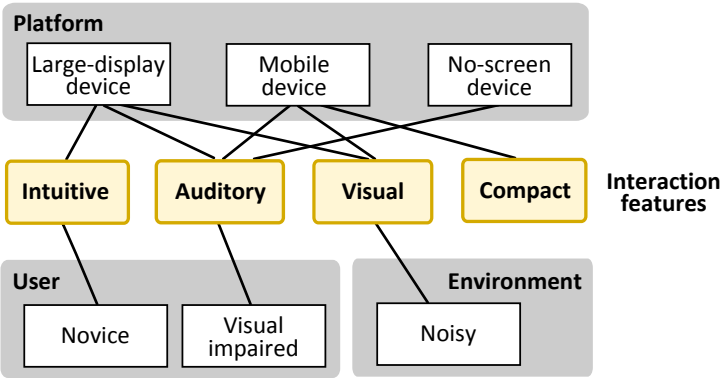

Figure 4.13: Decomposition of context conditions

ables the interaction to be context-aware. When a context change is performed, interaction is migrated to fulfill the requirements for the new context. For example, if a noisy environment is entered by the user, the auditory features are replaced by other features that are not auditory. This is because the noisy environment does not allow auditory interaction.

5. Selecting the interaction technique

Designers must define the appropiate interaction technique for each obtrusiveness level. This is done through the mapping between each fragment in the obtrusiveness space into a set of interaction features representing interaction mechanisms available. These set of features are the interaction aspects preferred for a specific obtrusiveness level. Note that the set of interaction features selected must fulfill the constraints among them represented by their relationships.

Figure 4.14 shows an example of the selection of the adequate interaction features for the obtrusiveness levels. For example, when a service is in the *proactive-aware* space, interaction is offered in a *graphical* and *speech* manner and the feedback is *change-based* which means that only the changes are reported (these features are activated for this

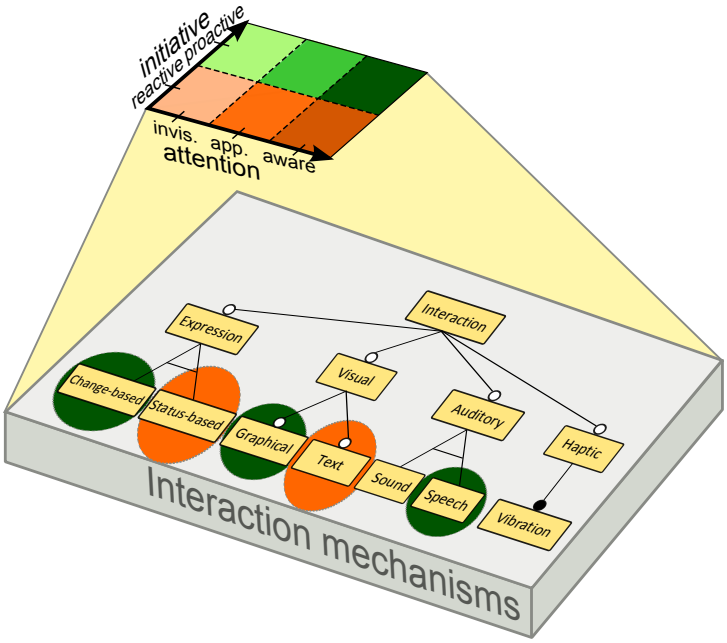

Figure 4.14: Selecting the interaction feature for an obtrusiveness level.

obtrusiveness aspect).

6. Composing concrete components

In this step, designers have to define the concrete user interface components that support the interaction techniques available defined by features (previous step).

For representing the concrete user interface components, we assume a user interface model that is organized in a **tree structure**. In this structure, components can be contained in other components following a hierarchical representation that allows an easier definition of UIs and an easier support for animation, multitouch interaction and visual effects. This interface model can be used for any platform that has a node-

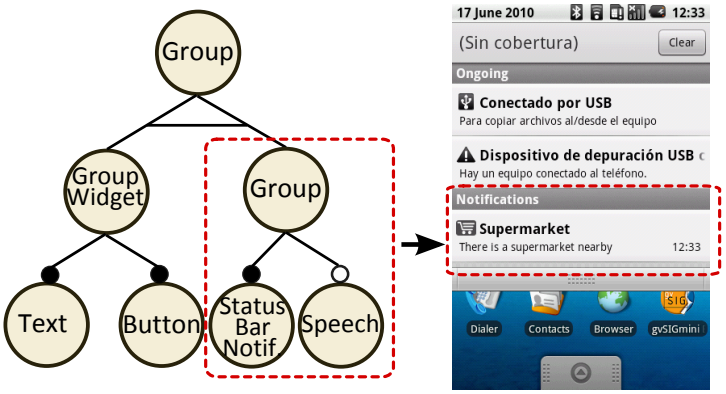

Figure 4.15: Concrete Interface model of the "Supermarket Notification"

based user interfaces such as Android, JavaFX[1] or the iPhone. This node-based user interface model provides an easier node substitution (to adapt UIs at run-time) and an advanced management of interaction events.

In our work, the nodes represent concrete interaction objects. They are any UI component that the user can perceive such as graphical objects, text, image viewers, UI controls, widgets, video viewers, etc.

An example of the concrete user interface model is shown in Fig. 4.15. This example shows the user interface components for a supermarket notification. Depending on the user needs and preferences, the notification will be shown by either a *widget* (left branch) or a *status bar notification* (right branch). If the *status bar notification* is chosen, *speech* interaction could be used at the same time (it has an optional constraint). The final user interface corresponding to the right branch of the tree is shown at the right of the figure.

We have taken from the notation of Feature Models the relationships (optional, mandatory, etc.) to indicate the constraints between the nodes. The constraints defined on them determine when they can be enabled or disabled according to the resource availability and the

[1] http://javafx.com/

interaction features activated.

In our approach we do not impose any particular semantics for nodes and the relationships represented in the model. For example, nodes can be visual elements and relationships can be visual containment relationships but it is not mandatory. The particular semantics depend on the final interaction components used.

Is worth noting that when a node is activated, all the constraints defined in the model must be fulfilled. For example, the activation of a node can trigger the deactivation of a sub-tree of the model. In order to determine the impact of a change in the subset of active nodes, analysis tools such as FAMA (Benavides et al., 2005) can be used. FAMA provides analysis capabilities to determine whether a selection of nodes from a feature model is consistent with the constraints defined and propose possible changes in the case some constraint is no fulfilled.

Linking interaction to components

Designers must define how each feature in the interaction model is specified in the concrete interface model. To achieve this, each feature is mapped into a set of nodes representing concrete interaction objects. This determines which UI components must be used to support each interaction technique in a concrete manner. This model also allows the automatic generation of user interfaces for a concrete platform.

In this way, when an interaction mechanism will be activated for a given service, corresponding concrete UI components will be activated too.

Figure 4.16 shows the mapping between the interaction features and the concrete user interface components. In this case, the interaction should be produced in a *status-based text* manner (these features are activated in the Feature Model). For these features, the corresponding nodes in the concrete UI model will be activated. In particular for this concrete example, it is activated the *Group* node that contains the *Status Bar Notif.* node to support the *change-based* and *graphical* interaction and the *Speech* node to obtain auditory feedback.

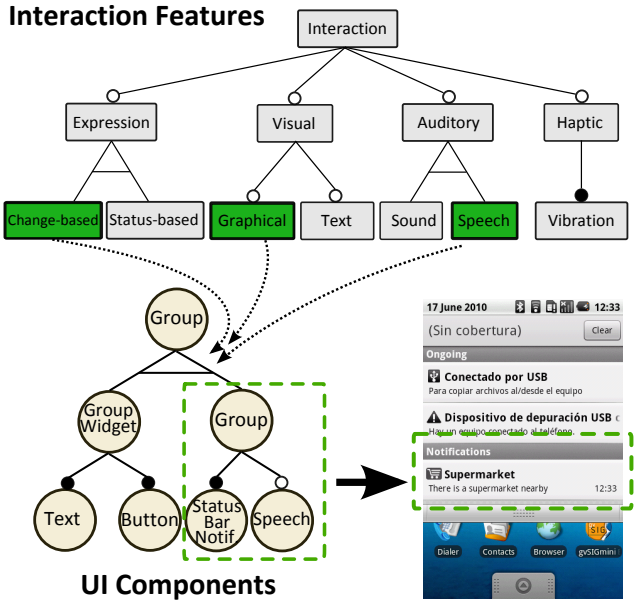

Figure 4.16: Mappings between interaction features and concrete components

4.2.3 Architecture description

The last step in the design phase is to describe the architecture components of the system (see Figure 4.17).

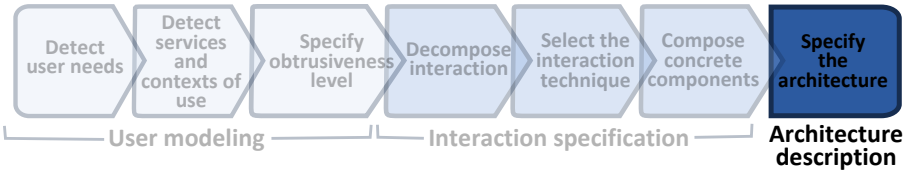

Figure 4.17: The task involved in the architecture description stage.

The modeling primitives introduced in the section above allow the specification of unobtrusive services where user attention is the main

resource that we take into account. Using these primitives, requirements for determine the adequate interaction mechanisms in terms of obtrusiveness are defined in an abstract manner and the concrete components for a specific platform are specified. In order to determine the way these components are integrated with the other components of the system, designers have to describe the component architecture of the whole system.

This architecture is also captured by means of models in order to (1) explicitly state their rationale and (2) to use this knowledge for automating the development in later steps. Further detail of this description is provided below.

7. Specifying the architecture

With this model, *designers* define the way user interfaces of the different services are integrated with the different components from the current and external systems. This allows to express the dependencies of an application in terms of data and functionality and detect the sources of context information that can trigger user interface adaptation.

In order to do so, the components of the specific mobile platform used for the system are captured in a model. In this case, we have defined a Domain Specific Language (DSL) (van Deursen et al., 2000) to capture the components from the Android application framework. A simple notation has been designed to represent the Android components and their communication mechanisms. This notation uses concepts that are familiar to Android developers in order to describe the setting in which the user interfaces take place. When the approach is applied to a different platform a new DSL must be defined.

We have chosen Android to apply our approach since adaptation plays a key role in this platform. Android applications should consider (1) different kind of context conditions, and (2) different hardware configurations. On the one hand, much of the available Android devices are capable of determining context information such as the user location and orientation. On the other hand, Android devices of different kinds are available today including mobile phones, netbooks or e-book

readers. In addition, these devices are usually provided with different screen sizes and input (e.g., physical or virtual keyboard).

Using this model, we focus on the general components defined for a mobile architecture, instead of dealing with technical implementation details of the platform. Some of these components are similar to components defined by traditional software architectures. For example, the notion of *Service* and *Repository* used in Android are the same than the described by the Domain Driven Design (Evans, 2003). Additionally, another specific components are included to support a mobile architecture such as *intents*.

The Android platform provides loosely-coupled components such as *Activity*, *Service*, *Content Provider* and *Broadcast Receiver*. An *Activity* presents a visual user interface designed around a well-defined purpose (e.g., viewing, editing, dialing the phone, taking a photo, etc.). A *Service* provides functionality that is executed in the background (e.g., a service that plays music). A *Content Provider* makes data available to other applications and a *Broadcast Receiver* is a component that reacts to announcements from other components. Broadcasts can originate from system code (e.g., indicate that the battery is low) or other applications and they are useful to support reactive behavior. The communication mechanism defined among Android components is based on *Intents*. An *Intent* is an abstract description of a desired action (e.g., obtaining an image) regardless of the component that provides this functionality. The intent mechanism allows components from different applications to integrate their functionality in an open manner.

Figure 4.18 illustrates the notation used to describe Android components. The main components from the Android application framework are represented in a graphical manner. The notation is aligned with other common notations such as Business Process Management Notation (BPMN) (OMG, 2006) or the Unified Modeling Language (UML) (Rumbaugh et al., 1998) for the sake of intuitiveness.

The intent-based communication mechanism among components is also represented in our notation (see Fig. 4.18, right) to indicate their possibilities for the components to interact. The way in which compo-

Figure 4.18: Graphical notation used to represent components of the Android application framework

nents handle intents is depicted in a different manner depending whether we are describing the capabilities of a component to either launch or receive a certain kind of intent. When the broadcast mechanism is used, the previous notations are slightly modified to indicate so. The capability of a component to launch an intent is depicted by means of an arrow. If the broadcast mechanism is used, the arrow is decorated with an asterisk to indicate that it can reach multiple receivers. Since intents are used as abstract descriptions of an action, the target component is not always known at design time. When an arrow connects two components, it describes an explicit intent. However, arrows are not forced to be connectec with a target element. In order to indicate that a component can respond to a given intent we make use of the lollypop primitive (used in UML for declaring an exported interface). When the component is a broadcast receiver the lollypop is decorated to resemble an antenna that can receive the broadcast.

Figure 4.19 shows the model for the components of a shopping list and supermarket notification services using the notation introduced. The system is composed by four activities corresponding to the user interfaces provided. These activities have defined the intent filters associated to the actions they can perform such as *ADD_ITEM* or *VIEW_ITEMS*. *Show Location* activity launches the intent *VIEW* to show the map of the location. Moreover, the *Show Services* activity has the intent filter *MAIN* to mark this activity as the initial activity. There are two content providers: one for offering the items of the shopping list and another for offering the information to update the *Widget Supermarket receiver*. There are also two services in the system: the *Shopping List* service in charge of orchestrating the communication be-

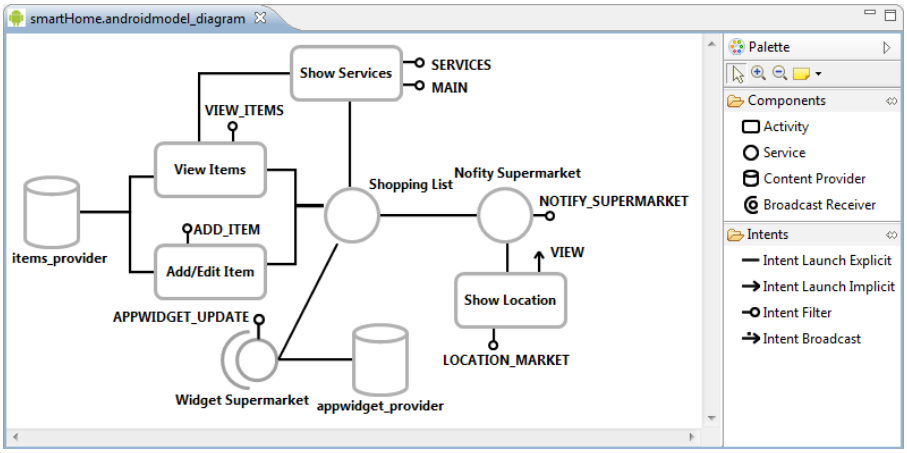

Figure 4.19: Component Architecture Model

tween the components and the *Notify Supermarket* service in charge of launching a notification.

On mobile platforms, such as Android, it is difficult to precisely determine the way in which the different interfaces are tight together just by observing the final user interfaces since different components influence in the user interface navigation. The introduced model captures relevant aspects for interaction such as (1) the components that require a user interface (i.e., Android Activities), (2) the possibilities for user navigation by means of intents, and (3) the different goals that each user interface must fulfill (e.g., add items or view items). Having these aspects separately, it is possible to define a combination of components for each user, personalizing the system to each user.

Although the approach has been applied to the Android platform, it has been designed to be general. Android-specific components are decoupled from adaptation aspects. Thus, a different component model (e.g., based on iPhone, Symbian, etc.) can be used instead without the need for redefining adaptation.

4.3 Tool support

The design method proposed in this chapter defines a set of concepts that are used to describe mobile service adaptation at different abstraction levels. Designers can combine these concepts to specify the architecture components of the application, the services offered by the system and the requirements for their interaction, the components of the concrete interface and the deployment configuration for supporting a given set of context conditions. A well-defined language must be used for this specification to avoid ambiguities.

This work follows a model-based approach for the development of mobile service adaptation. A model is a simplification of a system, built with an intended goal in mind, that should be able to answer questions in place of the actual system (Bézivin & Gerbé, 2001). Some examples of models are a scale plane in a wind tunnel, a plan of a house or a user interface described in a paper. In this case, we are dealing with descriptions of service interaction adaptation to guide their development. A modeling language can be defined as a set of models (Favre, 2004).

A metamodel is the description of a modeling language. A metamodel defines which models are part of this language. Plenty of models have been produced without metamodels or at least without making explicit metamodels (e.g., in the form of a hand drawing in a mat). Nevertheless, metamodels are useful to formalize and exchange models. By defining a metamodel that formalizes the concepts presented in our design method, we provide a clear rules about how to model service obtrusiveness adaptation according to our approach.

The modeling community has developed several projects to support the MDE paradigm under the Eclipse Modeling Project. Different tasks comprised by the MDE approach are supported: definition of a modeling language (metamodeling), description of a system using this language (modeling). For the implementation of the graphical tool we have used the possibilities offered by the Eclipse Graphical Modeling Framework (GMF) which is part of the Eclipse Modelling Project. GMF provides a generative component and runtime infrastructure for developing graphical editors based on EMF (Eclipse Modeling Framework).

This section formalizes the concepts used for describing unobtrusive mobile services. We have implemented a graphical editor tool that is based on Eclipse. Eclipse tools are defined by combining a set of plug-ins with different functionalities. We have developed some plug-ins to support the technique presented for the modeling of adaptation in mobile interaction, and we have integrated existing plug-ins that provide feature modeling capabilities that meet our requirements. These descriptions (expressed by means of models) facilitate the definition of dynamic aspects.

The following sections provide detail on the definition of the tools and the graphical environment. The derivation of a software solution that implements the requirements described by means of these models is detailed in Chapter 5

4.3.1 The obtrusiveness level metamodel

The obtrusiveness space metamodel defines the concepts used for describing the obtrusiveness level of a service. Figure 4.20 shows an excerpt of the metamodel. The *ObtrusivenessSpace* is composed by the *InitiativeLevel* and *AttentionLevel* elements that are the levels defined by the designers to describe the different degrees in which interaction with the system can be offered. *ObtrusivenessElement* is an abstract metaclass to represent the elements that can be mapped to the obtrusiveness space described. In this case, these elements are the services defined in the *Service* metaclass. On the one hand, a service can have several context conditions (*ContextCondition* element) and is associated to one or more Personas (*Persona* element).

An editor is provided to support the definition of the obtrusiveness space model using the EMF capabilities. Figure 4.21 shows the environment of the editor.

4.3.2 The Feature Model metamodel

In order to model the interaction mechanisms and the concrete user interfaces we have used Moskitt Feature Modeler (MFM). MFM is a

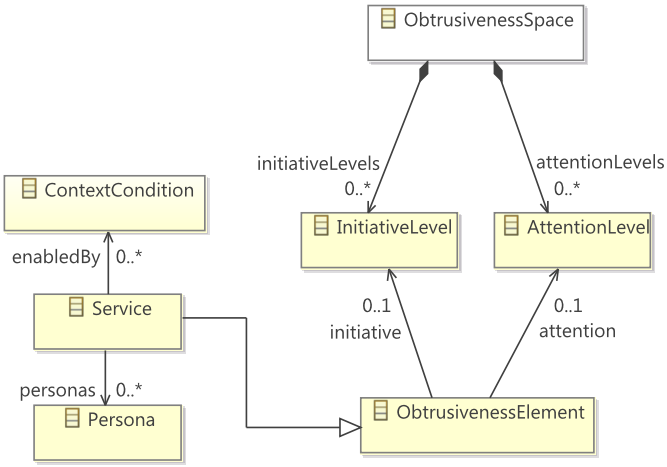

Figure 4.20: Obtrusiveness space metamodel

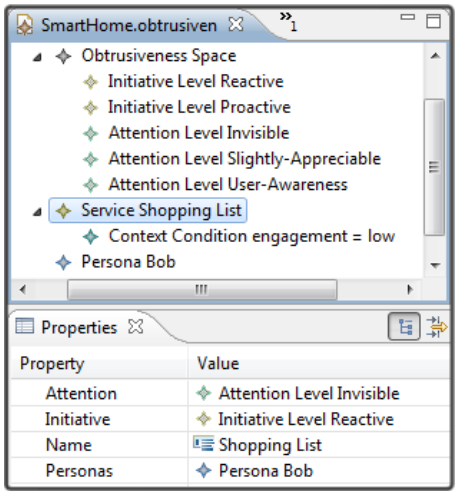

Figure 4.21: Obtrusiveness space editor

free open-source tool that is part of the Moskitt modeling suite[2]. MFM

[2]http://www.moskitt.org

is defined as a set of plug-ins that we could incorporate to enhance our tool support with feature modeling capabilities.

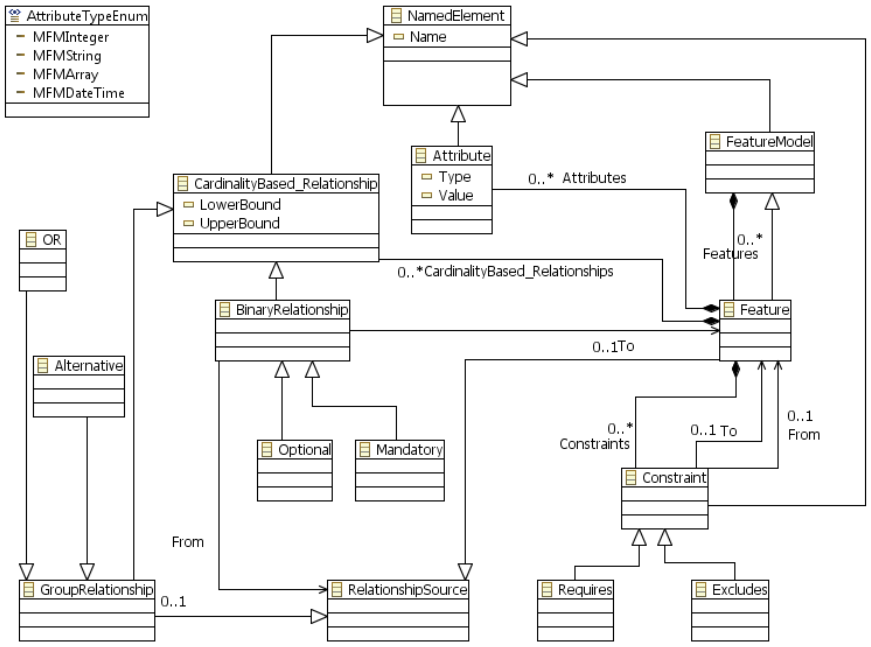

Figure 4.22: Feature Model metamodel

MFM provides features that are well suited for the use we are making of feature models. Figure 4.22 shows the different concepts in the Feature Model metamodel and the relationships among them. The metaclass *FeatureModel* normally is used as the root element of Feature models. The *Feature* metaelement represents the different features of the Feature model. Each feature can have attributes that are represented by the *Attribute* metaelement. Features are related among them through relationships represented by the *CardinalityBasedRelationship* metaelement. This metaelement is specialized in the different relationships that the Feature model supports: *Or*, *Alternative*, *Optional*, and

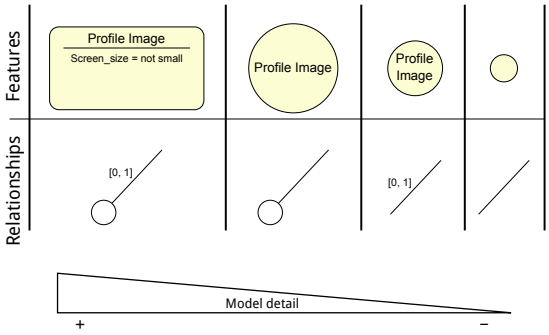

Figure 4.23: Different representations for interface nodes

Mandatory.

MFM is based on the generic formalization of the feature model syntax defined by Schobbens et al. (Schobbens et al., 2007). According with the results of their work, MFM incorporates support to multiple graphical notations. Users can dynamically **change the graphic notation** of feature models. This is very convenient when dealing with large user interface models (see Fig. 4.23).

MFM supports customizing the notation at any time between the following feature representations:

1. **Feature with Attributes.** Features are graphically represented by means of rectangles. These rectangles are composed of two compartments. The top compartment holds the feature name and the bottom compartment holds the features attributes. These features attributes follows the pattern: <name>:<type>=<value>.

2. **Rounded Feature.** Features are graphically represented by means of ellipses. The feature name is at the ellipse center, whereas feature attributes are not shown.

3. **Fixed Feature.** Features are represented as Rounded features which diameter depends of the feature name length.

4. **Simplified Feature.** Features are graphically represented by

means of ellipses. Neither the feature name is visible, nor the feature attributes. The ellipse diameter is set to a constant.

MFM also supports customizing relationship notation as follows:

1. **Cardinality-Graphic Relationship.** Relationships are represented by means of decorated lines and a flotating label. The line decoration indicates the type of relationship. Optional relationships are decorated with a white ellipse and mandatory relationships are decorated with a black ellipse. The label follows the pattern [min, max] to indicate the minimum and maximin cardinality of the relationship. Both label and decoration are synchronized between them.

2. **Graphic Relationship.** Relationships are represented by means of decorated lines.

3. **Cardinality Relationship.** Relationships are represented by means of lines and a flotating label.

4. **Simplified Relationship.** Relationships are represented only by means of lines.

By varying the representation a designer can go from a detailed description of the user interface nodes and their adaptation conditions to a general overview of the user interface topology. In addition, MFM allows to apply the different kinds of visualizations for different parts of the model. This resulted very useful for focusing on specific parts of the user interface while hiding the complexity for other parts.

Figure 4.24 shows the environment of the MFM.

4.3.3 The component architecture metamodel

The Android component metamodel defines in a formal manner the modeling concepts of the Android application framework. The metamodel is shown in Fig. 4.25. The metaclass *App* is the root element of

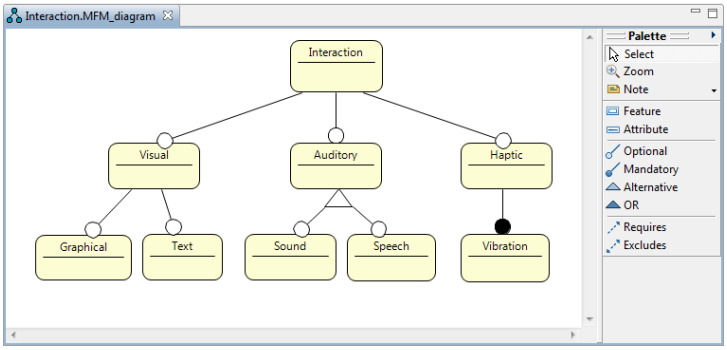

Figure 4.24: MFM environment

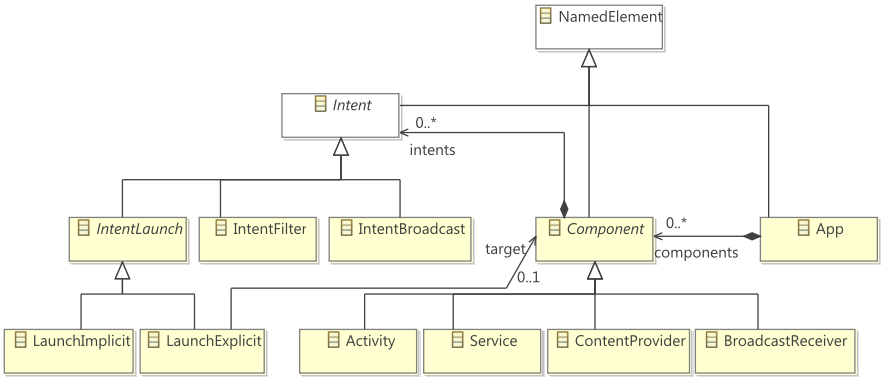

Figure 4.25: Android components metamodel

the component architecture model. The metamodel includes the definition of the Android components (*Activity*, *Service*, *ContentProvider*, and *BroadcastReceiver*) and the communication mechanisms among them (*LaunchImplicit*, *LaunghExplicit*, *IntentFilter*, and *IntentBroadcast*).

A graphical editor was provided for describing the components of different Android-based applications. As Fig. 4.26 shows, the developed tool incorporates a palette of Android components that can be labeled

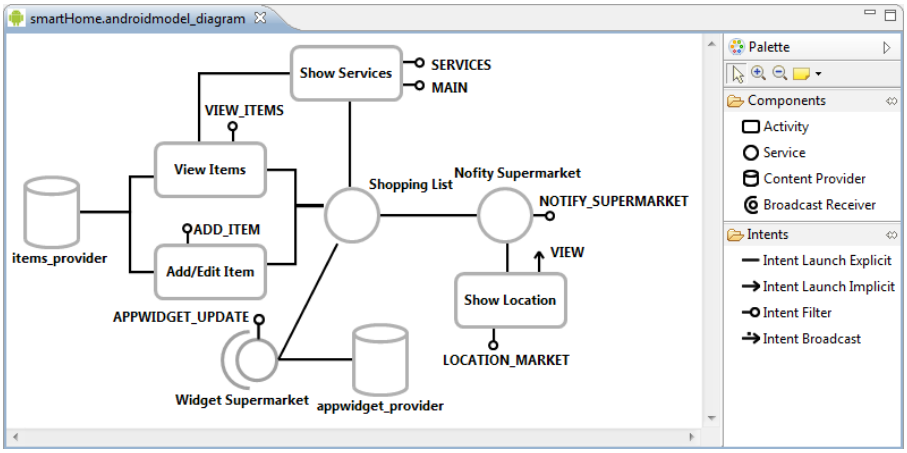

Figure 4.26: Graphical editor for Android components

and linked with other components. The components defined are the ones introduced in section 4.2.

With this tool developers can determine how an application interoperates with third party components (e.g., the contact list of the mobile device) or mock components defined to be later replaced with the final ones.

4.4 Conclusions

This chapter introduces a design method to specify services according to the user needs and context conditions in terms of obtrusiveness without duplicating efforts in the development. When defining mobile interaction many alternatives exist for defining the static structure of the user interface in a rapid manner (e.g., Visio stencils, paper prototypes). However, this only allows to validate the aspect of the UI in a particular moment. Our approach allows to visually represent how the interface changes according to different conditions. By analyzing the impact of different factors in the interface we can (1) produce specific

variants of the application to target a particular device kind, and (2) define how the interface is adapted at run-time acoording to different context conditions.

A DSL is defined in order to capture the architecture components required for a specific platform.

The design method provided relies on proven techniques and frameworks for context-aware modeling and implicit interaction design. The following chapter make use of the models defined to provide a development method for the unobtrusive mobile services.

Chapter 5

Prototyping and automating the development

The modeling primitives defined in Chapter 4 are useful for capturing user needs and interaction features for unobtrusive mobile services. The present chapter provides a method to validate the models defined and applies Model Driven Engineering (MDE) principles (Favre, 2004) to automate the development of context-aware mobile services in a systematic way. Thanks to the MDE techniques, it has been possible to traverse the gap between the high-level concepts used at design and the technical details of the particular mobile platform that is used for the system implementation.

The presented development process has been designed to allow a clear separation of concerns and minimize the impact of changes in requirements. On the one hand, the elaboration of the different models can be achieved by different development groups with different skills. On the other hand, the process follows an iterative and incremental approach for development, so the complexity of requirements changes can be handled effectively.

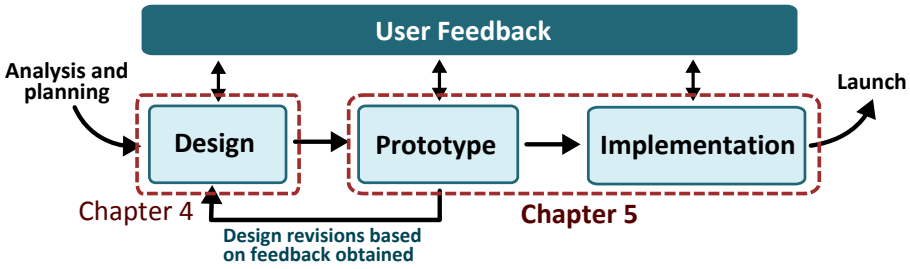

Figure 5.1: The stages proposed in the user-centered development process.

Fast-prototyping techniques are presented in order to evaluate the design and obtain feedback from end-users. Figure 5.1 shows an overview of the stages in the development process proposed introduced in Chapter 4. The unobtrusive mobile services are iteratively designed. The feedback obtained from the prototype is used to improve the original design. When no further changes are required, the system specification is used to guide the implementation of the final system. The design stage was introduced in Chapter 4. This chapter provides detail on the stages involved in the prototype and the implementation of unobtrusive mobile services.

The remainder of this chapter is structured as follows. Section 5.1 provides guidelines to validate in practice the designs obtained with the method. Section 5.2 describes the way in which the development process is automated. Finally, Section 5.3 concludes the chapter.

5.1 Prototyping to validate the design

The previous chapter introduced a design method for the definition of unobtrusive context-aware mobile services. However, when a context-aware system is designed, there is no guarantee that the resulting adaptation could met the user expectations. The User Centered Design cycle suggest to perform simulations of a scenario before it is finally implemented. Simulations rely on predefined tasks and generally use quan-

titative analytic methods including measuring aspects such as timing, error rates, and workload (Hagen et al., 2005).

This section provides techniques based on User Centered Design (UCD) for the evaluation of the impact for users of a context-aware mobile services when they are performed in the real world. De Sá and Carriço (de Sá & Carriço, 2006) showed that prototyping techniques can be determinant during the consequent evaluation stages, allowing users to freely interact with the system, improve them and use them on realistic settings without being misled. In this section we introduce a technique for the early-stage evaluation of mobile services by means of fast-prototyping. Our research results show that even through the proposed prototypes can be built quickly, they are capable of reproducing a level of user experience that is considered to be very close to what users expect from a final system. Thus, flaws in the adaptation design can be detected before efforts are put into the development of the final system.

5.1.1 Requirements for the evaluation

Researchers have shown that evaluating ubiquitous systems can be difficult (Neely et al., 2008). Many factors required for the evaluation of a system cannot be reproduced in a lab, but in-situ evaluation is also challenging and not feasible in many cases. Since a one-size-fits-all approach for evaluating ubiquitous systems is unrealistic (Neely et al., 2008), we have analyzed the specific application domain we are targeting and propose an evaluation model that fits the detected requirements. We detected the following challenges for the evaluation of unobtrusive context-aware mobile services:

Continuos evolution. If a change requirement is detected such as an interaction mechanisms for a task are not the most adequate, the system need to be changed. This evolution requires performing an evaluation of the system and the adaptation at an early stage of the development process that is both accurate (it provides a good estimation of the benefits to be obtained) and easily developed (it

requires minimal effort).

Concurrent environment. The evaluation of an ubiquitous system must take into account the integration with the rest of the activities that the user is involved in (Neely et al., 2008). This is especially relevant for mobility, where users can be interrupted and engaged in other tasks at the same time. Thus, a mobile service must be evaluated considering the interleaving tasks in which the user participates.

Physical conditions. Due to environmental changes, physical condition could get worse for users to perform an interaction (Yamabe & Takahashi, 2007). In mobile computing scenarios, users move around and the environment dynamically changes according to that. Thus, interaction should be evaluated taking into account these physical conditions.

Evaluation from the user perspective. Software usability and interaction adaptation are the key factors that determine the user experience in mobile user interface adaptation (Yamabe & Takahashi, 2007). These aspects are affected by different factors such as the kind of mobile device used or the way in which the service is presented. When evaluating interaction adaptation, both perspectives should be considered: user perception of the adaptability and the productivity increase for the system.

To fulfill the above requirements, we propose the use of early-stage prototypes. Even at early-stage prototypes, the need for more detailed and carefully built prototypes that offer resembling pictures of final solutions and their characteristics are suggested (de Sá & Carriço, 2009). These techniques enable iterative design, and provide frequent feedback about the potential of the designs.

5.1.2 Fast-prototyping for mobile service adaptation

The overall approach for the fast-prototyping of mobile service adaptation is illustrated in Figure 5.2. The goal of our evaluation method is

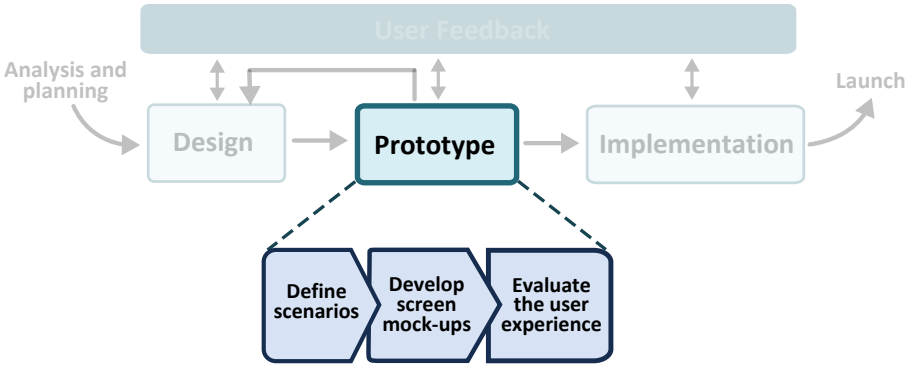

Figure 5.2: The different tasks in the evaluation method proposed

to immerse the user in an environment that makes the user feel as if he/she is using the final system despite the fact that a non-functional prototype is being used. The first step in the evaluation is to define a scenario according to the user needs and the adaptation that are being designed. The scenario should consider the concurrent nature and physical conditions of mobile environment. According to this scenario, users are provided a script to guide their actions.

An Android mock-up is designed for each user interface offered in the different contexts. These mock-ups provide the user with the expected interaction given a set of context conditions and user needs. Since the users have to follow a script that conforms a specific role, it is easy to anticipate the results that can be obtained.

The adaptation of interaction for each service is simulated using Wizard of Oz techniques (Dahlbäck et al., 1993). An operator provides the current user interface according to the context using another device. The operator is in charge of providing the correct user interface depending on the context of the user. In this way, the user is immersed in an environment that behaves like a working system with context-aware capabilities, but it is much easier to produce.

We have followed the steps described below in order to obtain fast-

prototypes for context-aware mobile services.

1. **Define a scenario where interaction adaptation is reflected.** A specific scenario is defined to illustrate the way interaction is adapted in terms of obtrusiveness. In the scenario, several services are presented at different obtrusiveness level depending on the user needs and context conditions. For example, we can define a video recorder service presented first in a *reactive-invisible* manner because it begins to record automatically in reaction to the user leave, and then the same service *proactively* notifies the user about to record the program in a *subtle* manner.

2. **Define the screen mock-ups for each user interface.** Depending on the obtrusiveness levels considered, the elements can be represented in a different manner according to the user needs (e.g., as taskbar notification). Links between screens must be provided in a decoupled manner. The screen to be shown depends on the current context of the user and its needs.

3. **Evaluate the user experience.** The screen mock-ups support the interaction that is required for the scenario defined. Evaluating the prototype composed by the mock-ups in naturalistic situations is essential to find key issues to be taken into account in the next iteration. The prototypes enable developers with a quick and inexpensive way to evaluate and assess the design ideas without implementing real and functional solutions.

For the development of Android mock-ups two opposed requirements are faced. We want mock-ups to be realistic but we also want them to be very easy to develop. To face both requirements we considered the use of a set of Android utilities for the development of Android GUI applications. There are different end-user tools to visually design the way the app looks instead of writing code. Some of these tools are the App Inventor for Android[1] and the DroidDraw UI Designer[2]. Using

[1] http://appinventor.googlelabs.com/about/
[2] http://www.droiddraw.org

these tools, we can quickly design UI layouts and the screen elements they contain, with a series of nested elements. In particular, Android has an XML-based layout file for each user interface. So, user interfaces can be easily defined by means of this layout file.

Specifically, to develop the Android mock-ups we have used the ADT Plugin for Eclipse that offers a layout preview for the XML file. Also, we used DroidDraw UI Designer to create the XML file since it is a graphical user interface builder for the Android platform which generates the XML files from UI designs. In particular, we used both utilities to achieve a look-and-feel for Android. Listing 5.1 shows an excerpt of an XML prototype developed.

Listing 5.1: Excerpt from the Android prototype.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android
      "
  android:orientation="vertical"
  android:layout_height="fill_parent"
  android:layout_width="fill_parent">

<TextView
  android:id="@+id/SupermarketName"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Mercadona"
  android:textStyle="bold"
  android:layout_marginLeft="10dip"
  android:layout_marginTop="5dip">
</TextView>
<TextView
  android:id="@+id/SupermarketAddress"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Calle del Gorgos, S/N, 46021 Valencia"
  android:layout_marginLeft="10dip">
</TextView>
<ImageView
  android:id="@+id/Map"
  android:layout_height="wrap_content"
  android:layout_width="wrap_content"
```

```
    android:layout_marginTop="15dip"
    android:scaleType="center"
    android:src="@drawable/mapa2">
  </ImageView>
  </LinearLayout>
```

The code illustrated above is a layout file that represents one screen mock-up. Each layout file must contain exactly one root element. Once the root element is defined, additional layout objects or widgets can be added as child elements to gradually build a View hierarchy that defines the layout. Figure 5.3 shows some mock-ups of the Android prototype. The rendering of the XML layout file showed above is shown in the top right of the figure.

Figure 5.3: Android prototype

Figure 5.4: Some screen mock-ups of the prototype

Another screen mock-ups of the prototype are shown in Figure 5.4.

The proposed technique makes it easy to apply the scenario defined in an environment that is close to the real one since it does not require much infrastructure. To apply this approach we only need WiFi connectivity, a mobile device and a physical environment that is similar to the real one.

In order to obtain valuable feedback from users a questionnaire has been used. The questionnaire uses questions from the IBM Post-Study questionnaire (Lewis, 1995) in conjunction with the questionnaire defined by Vastenburt et al. (Vastenburg et al., 2009). On the one hand, IBM Post-Study is a questionnaire that measure user satisfaction with system usability. On the other hand, some questions were taken from the Vastenburg questionnaire to evaluate messages acceptability and interaction adaptation. Also, we included a NASA task load index (TLX)[3] test. This test assesses the user's subjective experience of the overall workload and the factors that contribute to it. An example of the application of the evaluation technique can be found in Chapter 6.

[3]http://humansystems.arc.nasa.gov/groups/TLX/index.html

5.2 Automating the development

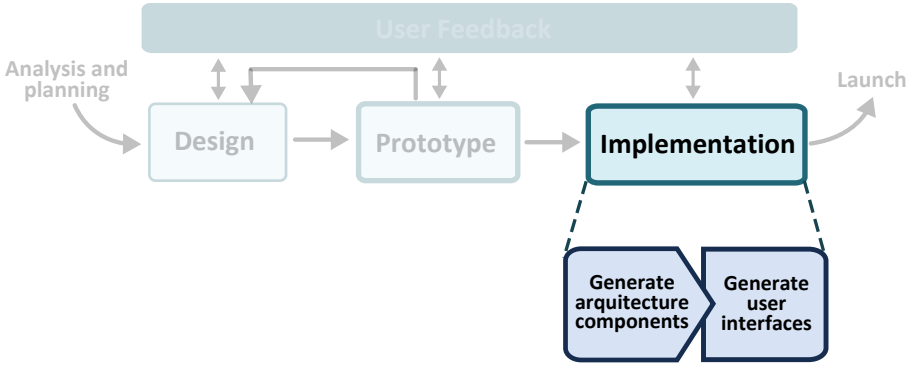

Figure 5.5: The different tasks in the implementation stage proposed

One of the main reasons for following a Model Driven Engineering (MDE) development is that it is focused on automation. Adaptation requirements change quite often, and systems need to evolve accordingly. By automating the development process, the system can adapt to requirement changes without losing quality. With the adequate tool support, changes in requirements can be mapped automatically to the particular technology the system relies on, facilitating its evolution.

Provided that modeling concepts are defined in a precise way, models can be transformed automatically into new models or code by means of model transformation techniques. This enables automation in system development since software artifacts can be derived in a systematic way. Many technologies and standards give support to this development paradigm. The Object Management Group (OMG) defined Model Driven Architecture (MDA) (Miller & Mukerji, 2003) to provide support to these ideas with standards for metamodeling and the definition of model transformations. Either following MDA or any other paradigm based on MDE ideas, software development can be improved by the raise in the abstraction level that the use of models provides. Figure 5.5 shows the tasks carried out in this implementation stage.

From the models obtained in the design stage we can generate the **architecture components** of the system and the **user intefaces**.

The following sections describe the definition of the architecture metamodel and the mechanisms applied for obtaining the architecture components of the system and the user interfaces of the software solution.

5.2.1 Architecture metamodel

MDE proposes the use of metamodels to formalize concepts and their relationships. A metamodel defines the constructs that can be used to describe systems. Using a metamodel, system descriptions become unambiguous at least at syntactic level. This makes the descriptions machine-processable.

The **metamodel of the system architecture** has been defined as the first step towards the automation of the development process. This metamodel captures the concepts defined in Chapter 4 such as the *Obtrusiveness Space* or the *interaction features*, and the constraints between them.

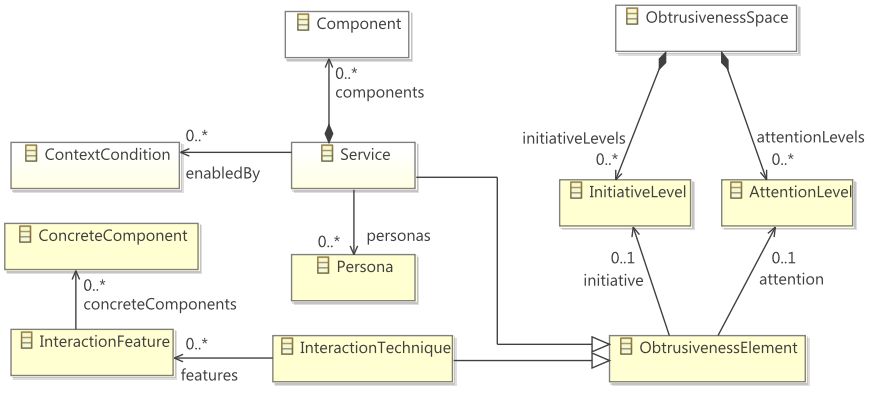

Figure 5.6: Excerpt from the system architecture metamodel

Figure 5.6 shows a diagram for the system architecture metamodel. The metaelement *ObtrusivenessSpace* is composed by the *InitiativeLevel* and *AttentionLevel* elements that are the levels defined by the designers to describe the different degrees in which interaction with the system can be offered. *ObtrusivenessElement* is an abstract metaclass to represent the elements that can be mapped to the obtrusiveness space described. In particular, these elements can be services defined in the *Service* metaclass and interaction mechanisms defined in the *InteractionTechnique* metaclass. On the one hand, a service is defined for one or several *Personas*, it can be composed by different components (*Component* element) and it can have several context conditions (*ContextCondition* element). The *Component* element extend the *Component* element from the Android components metamodel. On the other hand, an interaction technique can be related to several interaction features from the feature model of interaction mechanisms. This is represented by the element *InteractionFeature*. Note that this element extend the *Feature* element from the Feature Model metamodel in order to allow other elements in the architecture metamodel to be linked to them. These interaction features can be linked to several concrete nodes (*ConcreteComponent* element) that represent the concrete interaction mechanisms. This element extend also the *Feature* element because we have used the notation of feature models to represent the nodes.

By using the constructs defined in the metamodel, different system adaptations can be modeled. Since the concepts used for this description have been formalized, the resulting models can be processed automatically by different MDE tools. For the definition of the architecture metamodel, concepts have been formalized using Ecore. Ecore, part of the Eclipse Modeling Framework[4] (EMF), is a language targeted at the definition of metamodels with precise semantics. EMF provides tool support for the definition of metamodels and the edition of models.

We have defined our metamodel as the first step towards the automation of the development process. The use of EMF enables metamodels to be machine-processable. This allows other EMF-compliant tools to

[4]http://www.eclipse.org/modeling/emf

manipulate the specifications with different purposes (check properties, define graphical editors for the specification, etc.). In particular, we make use of code generation techniques in this work to automate the development as it is illustrated in the following sections.

5.2.2 Glue code generation

The development process generally involves several repetitive tasks. For our target technology, the definition of each *Component* involves actions like the definition of an *Android Activity* to produce the user interface and the definition of the component in the *Android Manifest* configuration file. This boilerplate code can be automatically generated by the information captured in system models. In this way, developers can focus on implementing only relevant business-logic.

We provide code generation capabilities for the development method described in the present work. This development considers Android as the target technology, but the approach followed allows developers to define different mappings to target other technological platforms.

From the description of a system based on the defined metamodel, source code can be generated with model-to-text transformation techniques. Model-to-text generation tools provide mechanisms to traverse models and generate the code associated with them. We applied model-to-text transformations to formalize the development process defined in Section 4.1. Glue code generation has been implemented using XPand templates from the Model-to-Text (M2T) project[5], which is part of the Eclipse Modeling Project. The application of templates to models is similar to the way templates are used to generate dynamic web pages in the web application development area. Model elements can be iterated and pieces of code can be produced instantiating them with values obtained from the model. XPand is a statically-typed template language with several features that simplify the code generation:

Polymorphic template invocation. Inheritance relationships in the source metamodels can be leveraged when templates are defined.

[5] http://www.eclipse.org/modeling/m2t

Given a set of modeling elements that are involved in inheritance hierarchy, specific behaviors can be easily defined for the different sub-types. When multiple templates are available for an element, the code generation engine applies the template variant that is more specific to the current kind of element.

Functional extensions. Metamodels can be extended in a non obtrusive manner to obtain derived information easily. This information is accessed as if it were part of the metamodel. However, these extensions do not affect the metamodel since they are only accessible during the transformation. Thus, generation rules are more readable and less dependent on the metamodel structure, which improves the generator maintenance.

A flexible type system abstraction. XPand provides support for some built-in types including simple types (String, Boolean, Integer, and Real) and collections (List and Set). In addition to built-in types, the type system can be extended with the concepts defined in the different metamodels.

Model transformation and validation facilities. In order to ensure that the models that are used for the generation meet certain conditions, they can be analyzed prior to the transformation is applied. By validating the input, we can ensure that the generator does not find unexpected information (e.g., components with the same name that would lead to a nameclash when code is generated). Furthermore, facilities are provided to transform these models in order to fix the problems detected.

The following listing 5.2 shows a general structure of template files in order to introduce the basics of the XPAND language. A template file consists of any number of *IMPORT* statements, followed by any number of *EXTENSION* statements, followed by one or more *DEFINE* blocks (called definitions).

Listing 5.2: General structure of a template file.

```
«IMPORT meta::model»
```

```
«EXTENSION my::ExtensionFile»
«DEFINE javaClass FOR Entity»
   «FILE fileName()»
      package «javaPackage()»;

      public class «name» {
         // implementation
    }
   «ENDFILE»
«ENDDEFINE
```

The basic statements used in the XPAND language are:

IMPORT. This statement is used to import a namespace in order to use the unqualified names of all types and template files contained in that namespace.

EXTENSION. Extensions provide a flexible and convenient way of defining additional features of metaclasses. For example, it is used to specify additional behavior such as query operations, derived properties, etc.

DEFINE. The *DEFINE* block is the smallest identifiable unit in a template file. By means of the *DEFINE* statement we can declare de rules in our template.

FILE. The *FILE* statement defines the output file for the code generation.

EXPAND. The *EXPAND* statement "expands" another *DEFINE* block (in a separate variable context), inserts its output at the current location and continues with the next statement. This is similar in concept to a subroutine call.

Iterators. XPAND use iterators primarily for iterating collections of model elements from a source model. The iterators available are: FOR, FOREACH, and IF.

The current implementation provides code-generation capabilities for two different aspects: (1) the architecture components for the system, including the *Android Manifest* and the different *Android classes* that are required for the implementation of the different components from the component architecture model and (2) the different user intefaces for a specific configuration. A detailed description of the artifacts generated is provided below.

1. Android components

The first step in the development of the system is to **generate the Android components of the whole application** defined from the component architecture model. Figure 5.7 describes the elements that are produced by the transformation from the model.

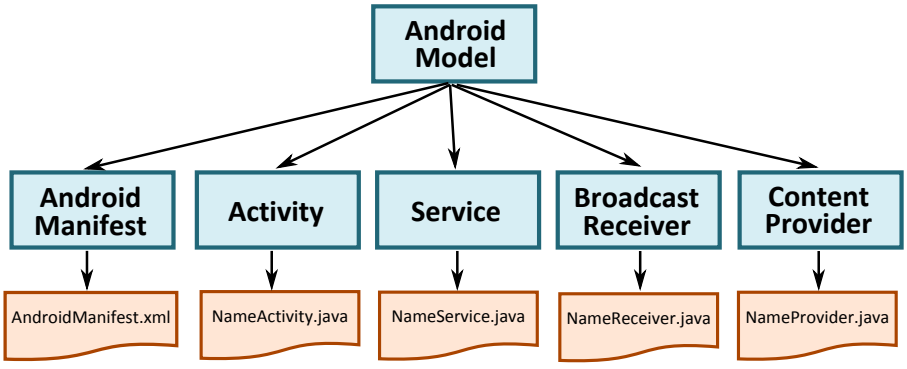

Figure 5.7: Global schema of the elements generated by the transformation.

The listing 5.3 shows the code of the main template that is used for orchestrating the generation of all the components. The excerpt of the transformation declares the *main* rule by means of the *DEFINE* keyword. The *main* transformation rule is used to generate the components of the application. In the example, we show the rule that is applied to the whole system. The rule named *main* is defined for the

App element of the Android components metamodel (see the metamodel in Section 4.3). By means of the *EXPAND* command, the initial structure of the application, the *Android Manifest* file and the rest of Android components are expanded for the generation.

Listing 5.3: Excerpt of the code generation template that produces the calls to all the components of an Android application.

```
«IMPORT AndroidModel»
«EXTENSION template::GeneratorExtensions»
«DEFINE main FOR App»
  «EXPAND template::InitialStructure::initialStructure»
  «EXPAND template::Manifest::manifest»
  «EXPAND template::Activities::activities»
  «EXPAND template::Services::services»
  «EXPAND template::BroadcastReceivers::broadcastReceivers»
  «EXPAND template::ContentProviders::providers»
«ENDDEFINE»
```

The listing 5.4 shows the code of one of the templates that is used for generating the *Android Manifest*. The excerpt of the transformation declares the *manifest* rule. The manifest transformation rule is used to generate the fragment of the *Android Manifest* that is associated to each component. In the example, we show that this rule is also applied to the whole system since it is defined for the *App* element of the Android components metamodel. Generation rules control the creation of new files (e.g., source code, configuration files, resource descriptions, etc.) and the generation of their correspondent content. The *FILE* statement defines the output file for the code generation (in the example, an *AndroidManifest.xml* file is generated into a folder named after the *App*).

Listing 5.4: Excerpt of the code generation template that produces the Android Manifest file.

```
«IMPORT AndroidModel»
«EXTENSION template::GeneratorExtensions»
«DEFINE manifest FOR App»
«FILE name + "/AndroidManifest.xml"»
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res
    /android"
```

```
        package="«generatePackageName(name)»"
        android:versionCode="1"
        android:versionName="1.0">
      <application android:label="@string/app_name">
          «EXPAND activitiesDeclaration FOREACH activities()»
          «EXPAND servicesDeclaration FOREACH services()»
          «EXPAND broadcastReceiversDeclaration FOREACH
              broadcastReceivers()»
          «EXPAND providersDeclaration (name) FOREACH
              providers()»
      </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
«ENDFILE»
«ENDDEFINE»
```

The rest of the rule is a code template with static and dynamic parts. The static parts of code are transferred to the generated code directly. In the example template, the static parts represent aspects that are common to any Android manifest, such as the XML header or the application declaration. The dynamic code is calculated for each instance to which the rule is applied. Dynamic expressions (defined between angle quotes) are used for capturing the required information and expressing it according to the target technology. For example, the package name for the *Android Manifest* is obtained from an auxiliary function *generatePackageName* that is defined as an extension of the metamodel. The extension statement in the example is in charge of importing the *extension* of the metamodel that implements the *generatePackageName* operation.

The generation of the rest of the *Android Manifest* is carried out by different rules, each one for a specific kind of component. For example, *activitiesDeclaration* rule is applied to *Activity* components for the generation of the activity statements. Listing 5.5 illustrates the definition of the *activitiesDeclaration* rule applied to *Activity* components. This rule declares the activity in the manifest and their intent filters according to the model defined. For each intent filter declared in the model, the rule adds an intent filter named with the name declared in the model. If the component has the Main intent filter, the default

category *android.intent.category.LAUNCHER* is also added. This determines that the activity can be launched by the user directly. The template of this example makes use of conditional statements (see the IF, ENDIF instructions) to add the default category for the generated intent.

Listing 5.5: Generation rule that produces the Android Manifest fragment corresponding to Activities.

```
«DEFINE activitiesDeclaration FOR Activity»
  <activity android:name="«generateClassName(name, "
      Activity")»">
    «EXPAND intentFilters FOR this»
  </activity>
«ENDDEFINE»

«DEFINE intentFilters FOR Component»
  «IF intents.typeSelect(IntentFilter).size > 0»
    <intent-filter>
    «FOREACH intents.typeSelect(IntentFilter) AS
        intentFilter»
      <action android:name="«intentFilter.name»" />
      «IF intentFilter.name == "android.intent.action.MAIN"
          »
      <category android:name="android.intent.category.
          LAUNCHER" />
      «ENDIF»
    «ENDFOREACH-»
    </intent-filter>
  «ENDIF»
«ENDDEFINE»
```

The code generation supported in this part automates the definition of the **Android Manifest** and the **Java classes** that are required for the implementation of the different components according to the models defined in Section 4.1. **Intent processing code** is also generated. Although full code generation is not provided for component implementation, the provided code skeletons let developers focus on the implementation of the business-logic behavior, avoiding to deal with particular details of the target technology. Since the Android specific artifacts are generated, the user of Android application framework is

made transparent to the developer, who only has to deal with Java programming.

2. Android user interfaces

The last step to complete the development of the system is to generate the user interfaces for a specific configuration of context conditions.

As we introduced in Section 4.1, Android platform has a node-based user interface. In particular, in an Android application, the user interface is built using *View* and *ViewGroup* objects. *View* objects are the basic units of user interface expression on the Android platform that serves as the base for subclasses such as *widgets*. The *ViewGroup* class serves as the base for subclasses called *layouts* which offer different kinds of layout architecture, like linear, tabular and relative. A *View* object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen. A *View* object handles its own measurement, layout, drawing, focus change, scrolling, and key/gesture interactions for the rectangular area of the screen in which it resides. As an object in the user interface, a *View* is also a point of interaction for the user and the receiver of the interaction events.

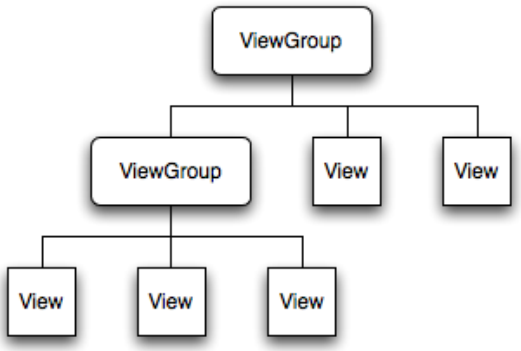

Figure 5.8: Hierarchy for defining Android UIs

Thus, on the Android platform, user interface is defined using a hierarchy of *View* and *ViewGroup* nodes, as shown in the Figure 5.8. The most common way to define a user interface expressing the view hierarchy is with an **XML layout file**. XML offers a human-readable structure for the layout, much like HTML. Each element in XML is either a *View* or *ViewGroup* object (or descendant thereof). *View* objects are leaves in the tree and *ViewGroup* objects are branches in the tree.

Thus, for the implementation of the user interfaces we produce the Android *XML layout file* for all the user interfaces of the whole system from the concrete UI model. But, in each generation we will produce the user interfaces of an specific configuration of context conditions. To define a configuration we use an auxiliary Configuration metamodel. The metamodel is shown in Figure 5.9

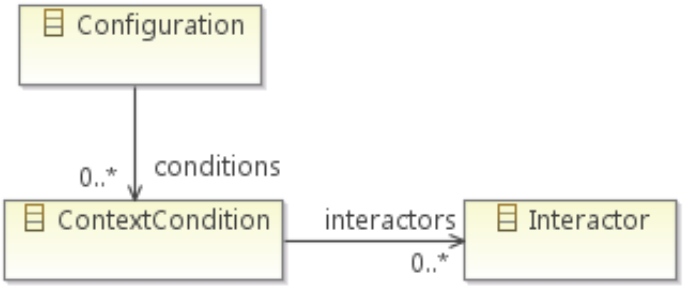

Figure 5.9: Configuration metamodel used to generate the interfaces.

In this way, we define a configuration that is composed by one or several context conditions that are active in a specific context. Each context conditions can have associated several interactors that correspond to the optional features in the Feature Model, since mandatory features are generated always.

The listing 5.6 shows an excerpt of the template used for generating the XML layout file. The transformation template is composed by

different rules that perform the generation. These rules are explained below:

Listing 5.6: Excerpt of the code generation template that produces the XML layout for each user interface.

```
«DEFINE Root FOR FeatureModelPackage::FeatureModel»
  «FILE Name+".xml"»
    <?xml version="1.0" encoding="utf-8"?>
  «EXPAND FindRoot FOREACH Features»
  «ENDFILE»
«ENDDEFINE»

«DEFINE FindRoot FOR FeatureModelPackage::Feature»
  «FOREACH Attributes AS e»
    «IF e.Name=="root" && e.Value=="true"»
      «EXPAND PrintFeature FOR this»
    «ENDIF»
  «ENDFOREACH»
«ENDDEFINE»

«DEFINE PrintFeature FOR Feature»
  <«Name»
  «FOREACH Attributes AS e»
    «IF e.Name == "root"»
      xmlns:android="http://schemas.android.com/apk/res/
          android"
    «ELSEIF»
      android:«e.Name»="«e.Value»"
    «ENDIF»
  «ENDFOREACH»
  >
  «FOREACH CardinalityBased_Relationships.typeSelect(
      Mandatory) AS e1»
    «EXPAND PrintFeature FOR e1.To»
  «ENDFOREACH»
  «IF ! CardinalityBased_Relationships.typeSelect(
      Alternative).isEmpty»
    «FOREACH CardinalityBased_Relationships.typeSelect(
        Optional) AS e1»
      «EXPAND PrintFeatureAlternative FOR e1.To»
    «ENDFOREACH»
  «ELSE»
  «FOREACH CardinalityBased_Relationships.typeSelect(
      Optional) AS e1»
```

```
      «EXPAND PrintFeatureOptional FOR e1.To»
    «ENDFOREACH»
    «ENDIF»
    </«Name»>
  «ENDDEFINE»
```

Root rule. The *Root* rule is applied to the whole concrete UI model (*FeatureModel* element). This rule is in charge of the creation of the XML file and the generation of the corresponding content. Then, the XML header is declared as a static part of the rule.

FindRoot rule. The *FindRoot* rule is applied to the different nodes (*Feature* element) for the generation of the rest of the XML layout. *FindRoot* is a recursive rule in charge of seeking the root node and calling the *PrintFeature* rule to begin the generation of the whole nodes of the hierarchy from the root node.

PrintFeature rule. *PrintFeature* rule generates the attributes for the nodes. Depending on the kind of node, the attributes to generate are different. If the current node to generate is the root node, the header of the layout is generated. Otherwise, the value of the attributes are generated. Then, depending on the variability relationships in which the nodes are linked, the appropiate PrintFeature rule is expanded (*PrintFeature*, *PrintFeatureAlternative*, *PrintFeatureOptional*). The generation of this alternative and optional nodes depend on the active context conditions defined in the *Configuration* model.

The advantage of declaring the UI in XML is that it enables to better separate the presentation of the application from the code that controls its behavior. UI descriptions are external to the application code, which means that it can be modified or adapted without having to modify the source code and recompile.

Furthermore, we generate code for a *status bar notification* since it can not be implemented by means of the layout file. The status bar notification is initiated from a *Service*. In this way, the notification

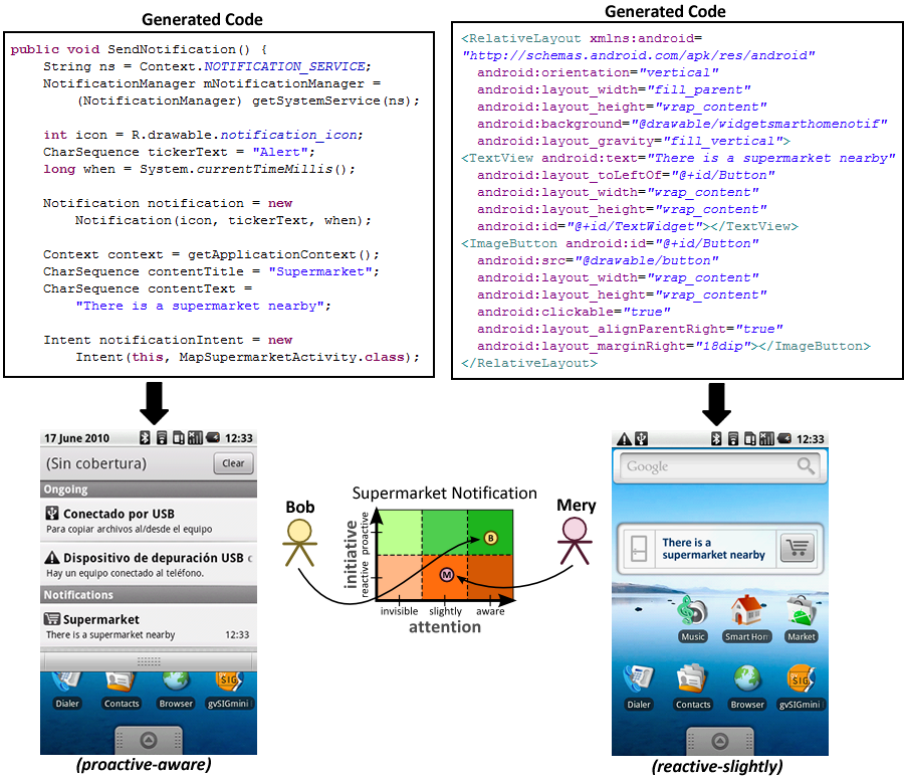

Figure 5.10: Different generations of the same service

can be created from the background, while the user is using another application. This is implemented checking the name of the node in the *PrintFeature* rule.

Figure 5.10 shows the generated code for the supermarket notification of two different personas. The service is in different obtrusiveness level for each persona, so the generated code is adapted according to the obtrusiveness level. On the one hand, for Bob (left branch), the service is in the *proactive-aware* obtrusiveness level. Through all the design process described in the previous chapter, the service is presented by

means of a *status bar notification* (see Fig. 4.15). An excerpt of the generated code for the *status bar notification* and the rendering of this code is shown at the left of the Figure. On the other hand, for Mery (right branch), the service is in the *reactive-slightly* obtrusiveness level because Mery wants to go to the supermarket without a notification. For this obtrusiveness level, a *widget* is used (see Fig. 4.16 to see the mappings of the models) and the generated code is shown at the right of the Figure. In this way, the services are generated and adapted for each persona.

5.2.3 Continuous evolution

Services in the obtrusiveness space could evolve due to changes in user needs or context conditions. This constitutes an evolution of the system in terms of obtrusiveness. Thanks to the decoupling role that the models play in the development process, this evolution is supported by the method in an easy manner. In this way, system maintenance is carried out at the modeling level.

First a change requirement is detected. This means that the obtrusiveness level for a service is not the most adequate. To validate the correctness of the specification, stakeholders are shown a prototype of the final user interface, which is more comprehensible for final users than the models used for the specification. Changes are commonly expressed in terms of the final interface since it is what the user perceives. Thus, the analyst needs to set up the properly obtrusiveness level for the service to fulfill the new requirements that satisfy the user.

Figure 5.11 shows an example of an evolution of the service. In the obtrusiveness space of context conditions we can see an event evolving from a reactive space to a proactive one (e.g., caused by a change in the user's location or user preferences). In this particular example, the notification of a supermarket nearby was in a *reactive-slightly* space because the user preferred to go to the supermarket without being notified explicitly. For this region in the obtrusiveness space, subtle interaction was preferred, activating the elements *status-based* and *text* and producing the final UI, showed at the left of the Fig. 5.11. But user

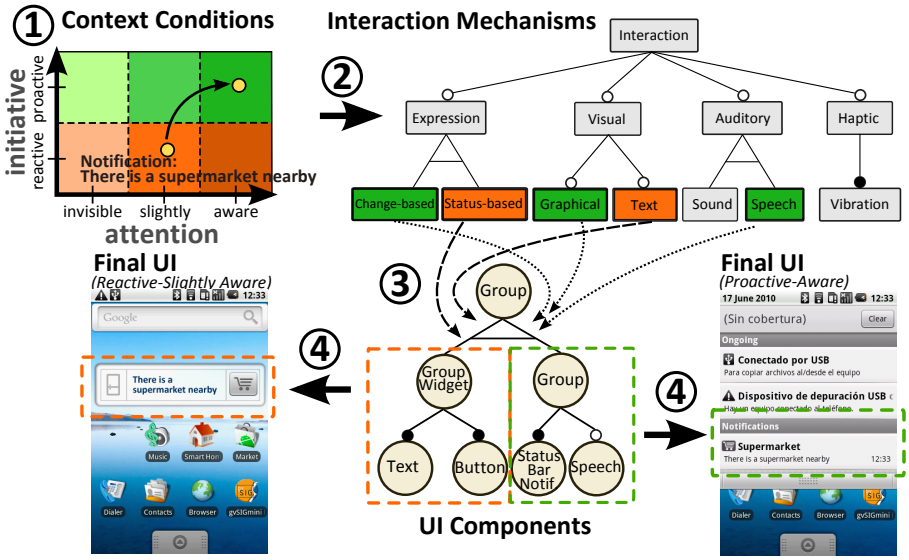

Figure 5.11: Service evolution

preferences or needs could change requiring another obtrusiveness level for the same service. In this example, user preferences changed and the user wanted to be notified when a supermarket was closer. Thus, this change in the obtrusiveness level required other interaction methods more explicit such as *change-based*, *graphical* and *speech*. Thus, another features was activated producing a different UI showed at the right of the figure.

5.3 Conclusions

This chapter provides a method to evaluate the designs and the mechanisms for automating the development of unobtrusive mobile services.

The designs defined according to the method can be easily put into practice. Fast prototyping techniques have been used to validate the adaptation for each service. The feedback obtained from the evaluation

can be used to better adjust the models defined at design-time.

By applying MDE principles, the requirements captured in the design stage are transformed into a final software solution. In this way, developers do not have to deal with technological details of the target platform. In addition, the definition of the architecture at modeling level, allows the presented approach to be sustainable since it can support the evolution of the system to new technologies.

Much of the complexity in software development is due to the rules that the different computing frameworks require but are not enforced by the programming language in use. Platforms such as Android, OSGi, Java ME, Enterprise Java Beans, or Google Web Toolkit, make use of the Java language but they require to follow different programming models that are not always simple to understand. By avoiding developers to deal with these technological constraints they can focus on business logic. Hiding this complexity is the main goal of our approach for the automation of unobtrusive mobile services. For example, the developers completing the generated code in our approach do not require to know how the intent mechanisms works in Android, since the intent processing code is already generated for them.

Chapter 6

Validation of the proposal

This chapter describes the application of our approach in practice. By applying our approach we want to illustrate how interaction can be adapted to provide an adequate obtrusiveness level in an Ambient Intelligence (AmI) environment. In particular, our method was applied in order to support different services in a smart home environment based on the scenario of service adaptation developed in (Cetina et al., 2009). We extended the services defined in the original case study in order to adapt the obtrusiveness level at which they are presented to the user. Based on this case study, we validate whether the method defined in this work can cope with the user attention requirements. In particular, in this chapter we are verifying the following aspects:

Design method. The information captured at design should describe the aspects that are relevant to capture obtrusiveness and interaction requirements. Our research results show that the models defined are useful for designers to discuss about attention and interaction requirements.

Iterative design. Requirements must be captured in a way that it is feasible to validate them with fast iterations. In this way, continuous feedback can be obtained to improve designs. Our research results show that fast-prototyping techniques can be applied in a way that reproduce the user experience of the final system.

The approach followed for validating the proposal is focused on the previous aspects. First, we wanted to verify that the design method was appropriate for describing the adaptation of interaction in pervasive mobile services in an unobtrusive way. In order to respond to this, we modeled the requirements for the services. Then, we make use of fast-prototyping techniques in order to re-design the process. On the one hand, feedback was gathered from end-users in order to verify how the design method could deal with the changes iteratively. On the other hand, we evaluated to which extend the prototypes provided are representative of a final system. This determined the usefulness of the fast-prototyping technique in providing quality feedback.

The remainder of this chapter is structured as follows. Section 6.1 introduces the case study and the design of the services involved. Section 6.2 describes the validation with end-users by means of early-stage prototypes. Finally, section 6.3 concludes the chapter.

6.1 Smart Home case study

The scenario of our case study describes a normal day in Bob's life and the way interaction mechanisms of different home services change depending on the context. Bob lives in a smart home with garden and a swimming pool. Every day, he gets up at 7 a.m. and drinks milk for breakfast while he watches a TV program before going to work. One day during breakfast, Bob runs out of milk. In reaction to this, the refrigerator added this item to the shopping list in an invisible manner for Bob. While he was watching the TV program, the system reminded him that he had an important meeting at work and he had to leave the house sooner. Therefore, the video service started to record it. Before leaving the house, he realized the pool was dirty

During the meeting, the smart home reminded Bob about watering the plants. Because of he had the mobile at hand, the notification appeared in a subtle manner suggesting him if he wanted that the system water the plants automatically. Meanwhile, he manually added some products to the shopping list using the mobile device.

When he was going back to home, he was nearby of a supermarket and the mobile notified him about it, showing the map to arrive to the supermarket. When he was there, the map was changed by the floor map of the supermarket. At the same time, the mobile suggested him the items to the shopping list that were available in that supermarket. While Bob was buying, the mobile suggested him a television series to record since he usually watched that program. When he arrived at home, he put the mobile to charge. While it was charging, pool was cleaned automatically. At the end of the day, Bob realized the pool was cleaned and the programs were recorded.

For the design of the services defined in the case study we applied the design method defined in Chapter 4. According to this method, different aspects of the services should be captured in models. The detail of the information captured in each modeling perspective is shown in Figure 6.1 through the different tasks. This tasks include the representation of the system's intended users thought *personas*, the *obtrusiveness level* for each service, the requirements for the *interaction* with the services, the *concrete interaction elements* that determine the final user interface provided, and the *architecture components* of the system. All these aspects are detailed below.

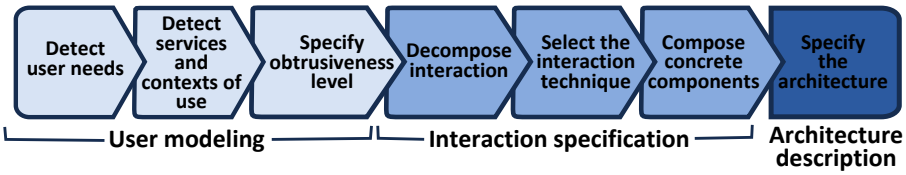

Figure 6.1: A detailed persona

6.1.1 User modeling

The goal of this stage is to understand the users and capture their needs and preferences. Moreover, services and context conditions are detected and services are specified in terms of obtrusiveness. These aspects are detailed below.

1. Detecting user needs

In order to give a clear picture of how users are likely to use the system and what they will expect from it we define personas. Personas capture relevant information about customers that directly impact the design process: user goals, scenarios, tasks, functionalities, and the like. Figure 6.2 shows the description of the persona for Bob.

2. Detecting services and contexts of use

After describing the persona and study their needs, the services defined for the Smart Home were the following:

Shopping list. Users are enabled to keep track of the products they want to buy in order to purchase these products on the next visit to the supermarket. The shopping list is shared among all the members of the house, including the smart refrigerator that can add items to the shopping list.

Video recorder. This service allows to record video in a digital format to a disk drive, USB flash drive, SD memory card or other mass storage device. Users can be asked to record a program or the program can be recorded automatically.

Agenda. It allows users to manage his time giving convenient access to their tasks alongside their calendar. Also, users are enabled to get event reminders when the task is going to begin.

Plant watering. Users have a busy lifestyle and they usually forget to water their plants. This service is in charge of remind and tell

Bob Berry · The busier

Familiar to Smart Home services

Behaviors

One channel — VENUE — Many channels

Low — ACTIVITY — High

One service — BREADTH — Many services

Objectives

· Optimize time
· Don't forget tasks
· Feel in control of housekeeping
· Keep track the items to buy
· Keep the house up-to-date
· Record favorite programs

Scenarios

· **Remember items to buy**
Bob lives alone and he has to take on all the responsabilities in the home. He always goes shopping and he never remembers what he need or the finished products. He wastes a lot of time buying because he has to move along the entire supermarket. He usually has a paper in the refrigerator to write down the items but he sometimes forgets to take the paper when he goes to the supermarket. Bob thinks he should keep track the items to buy in an electronic manner.
· **Recording interesting programs**
Bob usually watches a TV program before going to work when he is having breakfast. Sometimes he has a meeting and he has to leave home earlier loosing the program. He would like the program was recorded when he is not watching it.
· **Be aware of pending tasks**
Bob has a busy lifestyle and he sometimes forgets important tasks he has to do such as deadlines or meetings and other tasks that are less important but they are essential such as water the plants, birthdays, etc. He hopes be aware of pending tasks and events when it was required.
· **Optimize time**
Bob usually goes walking to the work. He passes in front of several supermarkets going back to home but he never remembers that he has items to buy and he has to return later. He wants to be aware that he has items to buy when he is nearby to the supermarket avoiding having to return later.

Concerns

· If a product of my refrigerator runs out, can the system add the item to the shopping list?
· How can I do not forget important tasks and events?
· If the system do something, will it inform me about it?
· If my favorite program begins and I can't watch it, can the system record it?
· How can I remember when water the plants?
· I am very busy. How can I make sure I am maintaing the house up-to-date?

Background

Bob is a single man who works in a big company and he lives alone in house with swimming pool. He has 32 years old. He works a lot because he wants to promote in the company. Thus, he has little free time. In his free time he likes practice sport and watching TV series and movies. He is very trendy and he has a smartphone. He likes new technology but he has few time to be up-to-date.

Figure 6.2: A detailed persona

users that their plants need water and water the plants automatically when they need it.

Supermarket notification. Many users do not remember that they have items to buy when they are nearby to the supermarket. So, the intention of this service is to prevent these situations, notifying users when they have items to buy in a nearby supermarket.

Clean pool. Swimming pool can be lots of work to keep clean. This service is in charge of swimming pool maintenance with the possibility of informing the users about the situation of the pool.

The contextual information relevant for the given persona is:

User location. A service can be adapted depending on the location of the user. For example, the supermarket notification service can change the information showed depending on this information. On the one hand, if the user is outside supermarket, the service can show the map to arrive to the supermarket. On the other hand, if the user is inside, the service can show the floor map to the supermarket.

Mobile location. The location of the mobile is another context factor that should be taken into account for Bob. Bob can have his mobile with him (e.g., in his hand, in his pocket) or the mobile can be far of Bob (e.g., if Bob leaves the mobile charging in other room). If Bob does not have the mobile with him, it is not necessary that services provide notifications by means of alarms avoiding disturb other people.

User engagement in other activities. This is an important factor to be taken into account for notifications. For example, if the user is engaged in an important activity, a notification will appear in a subtle manner avoiding to disturb him/her.

Once we have defined the personas, the tasks they can perform and the relevant contextual information in which the adaptation can depend on, we define the way in which tasks are presented in terms of obtrusiveness for the case study. This information is detailed below.

3. Specifying the obtrusiveness level

Each task can be provided at different obtrusiveness level depending on the user needs or the context conditions in which the tasks are performed. Figure 6.3 shows the tasks of the Smart Home case study and their obtrusiveness level for Bob. The obtrusiveness space in this case was defined by dividing each axis in different parts as it was illustrated in Chapter 4. The attention axe is divided in three levels depending whether the interaction should be *invisible* to the user, *slightly noticeable*, or *completely aware* for the user. The initiative axe is divided in two parts that represent interactions initiated by the user (*reactive*) and interactions initiated by the system (*proactive*).

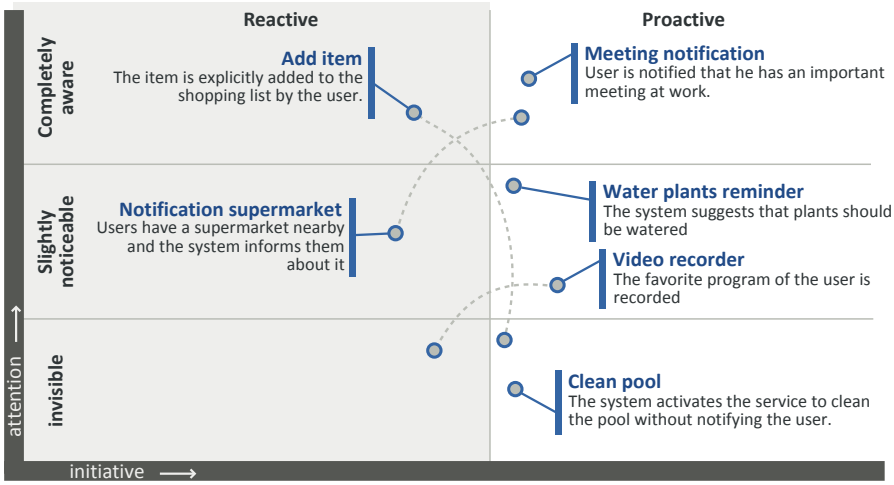

Figure 6.3: Obtrusiveness level defined for each service in the Smart Home case study.

During analysis we decided the appropriate obtrusiveness level for services according to the contextual information and the user needs. The obtrusiveness level for the different tasks in the Smart Home are detailed below.

Shopping list. The service to add an item to the shopping list can be placed in different obtrusiveness level. The item can be added to a shopping list explicitly by the user when he remembers an item to buy (*reactive and completely aware*) or it can be added by the system (*proactive*) when the user just drops the item to the garbage in an invisible manner (*invisible* level of attention).

Video recorder. The service to record a program is also offered in different obtrusiveness level. On the one hand, the system can begin to record the program automatically in an invisible manner for the user (*invisible* level of attention) in reaction to the user leave (*reactive*). On the other hand, the service can notify the user *proactively* about to record a TV program because the user usually watches the program. But the notification is only shown as a hint (*slightly noticeable*) because it is not very important for the user and when the notification appears in the case study Bob is shopping (engaged in other activity). Thus, it can be later implemented as a soft vibration or some non-intrusive mark on the screen to indicate that a notification exists.

Agenda. The notification of a meeting to the user is performed in a *proactive* manner in terms of initiative and the user is *completely aware* because the message is important for him.

Plant watering. Not all the information provided by the system is relevant for the user at anytime. Depending on the activities he is engaged in and the importance of the message, the user will prefer to be disturbed or not by the system. In this case, water the plants reminder is not very important for the user and he prefers to be notified in a subtle manner (*slightly noticeable*) by the system (*proactive*). Moreover, when the plants watering is notified Bob is engaged in other activities (He is in a meeting).

Supermarket notification. When the user is in the proximity of a supermarket (user location), he/she is notified about a supermarket nearby. Depending on the distance to the supermarket and the number of items on the shopping list, the notification will be

different. On the one hand, if the user is closer to the supermarket, the system *proactively* will notify the user in an explicit manner (the user is *completely aware*). On the other hand, if the user is far, the notification will be *slightly noticeable*. Then, when the user is in the supermarket, the system suggests the items to buy that are in the shopping list at the *slightly noticeable* level of attention.

Clean pool. The system *proactively* activates the service to clean the pool in an *invisible* manner for the user due to the user does not have the mobile nearby (mobile location).

In order to support the behavior described above for the tasks performed in the Smart Home case study, different interaction techniques can be applied. The mechanisms used from all the ones available for interacting with the system in the Smart Home are described below.

6.1.2 Interaction specification

The following step is to make use of the adequate interaction mechanisms to provide the functionality according to the obtrusiveness level. The tasks carried out in this stage are detailed below.

4. Decomposing the interaction

According to the previous requirements, different interaction techniques are used to provide the functionality of the services. This information is decomposed in a Feature Model in order to indicate the commonalities and differences between adaptation aspects and define the constraints that exist for the selection of the different features.

In (Chittaro, 2010), Chittaro summarizes the many channels that can be exploited to send and receive information from a mobile device. We have defined the feature model utilizing these modalities. For the constraints between modalities we have followed the study presented in (Lemmelä et al., 2008) to identify modalities and modality combina-

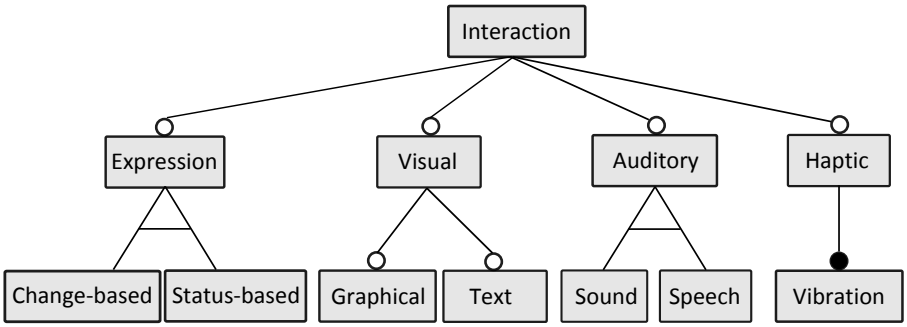

Figure 6.4: Decomposition of interaction aspects using the Feature Model.

tions best suited for different situations and information presentation needs.

Figure 6.4 shows the decomposition of available interaction in the Feature Model and the constrains for their selection. We have divided the interaction into groups of expression, visual, auditory and haptic modalities. These four main features include a set of manifestations of input and output modalities.

5. Selecting the interaction technique

Then, we have to choose for each task the interaction features that are going to support the obtrusiveness level defined. Table 6.1 shows a view of the interaction analysis results performed for the tasks. The table shows for each task, the obtrusiveness level at which it can be performed and the interaction features selected for the obtrusiveness level. The tasks that have two obtrusiveness level associated is because they depend on context conditions.

Task	Obtrusiveness	Interaction Features
Add item	(reactive, aware)	Text, graphical
Add item	(proactive, invisible)	-
Record program	(proactive, slightly)	Change-based, graphical, vibration
Record program	(reactive, invisible)	-
Meeting notification	(proactive, aware)	Change-based, graphical, speech
Water plants reminder	(proactive, slightly)	Change-based, graphical, vibration
Supermarket notification	(proactive, aware)	Change-based, graphical, sound
Items to buy suggestion	(reactive, slightly)	Status-based, text
Clean pool	(proactive, invisible)	-

Table 6.1: Interaction features for each task in the obtrusiveness space

6. Composing concrete components

In order to define the concrete user interface components that support the interaction techniques available we define the node tree of our system. For the Smart Home case study, the concrete components are shown in Figure 6.5.

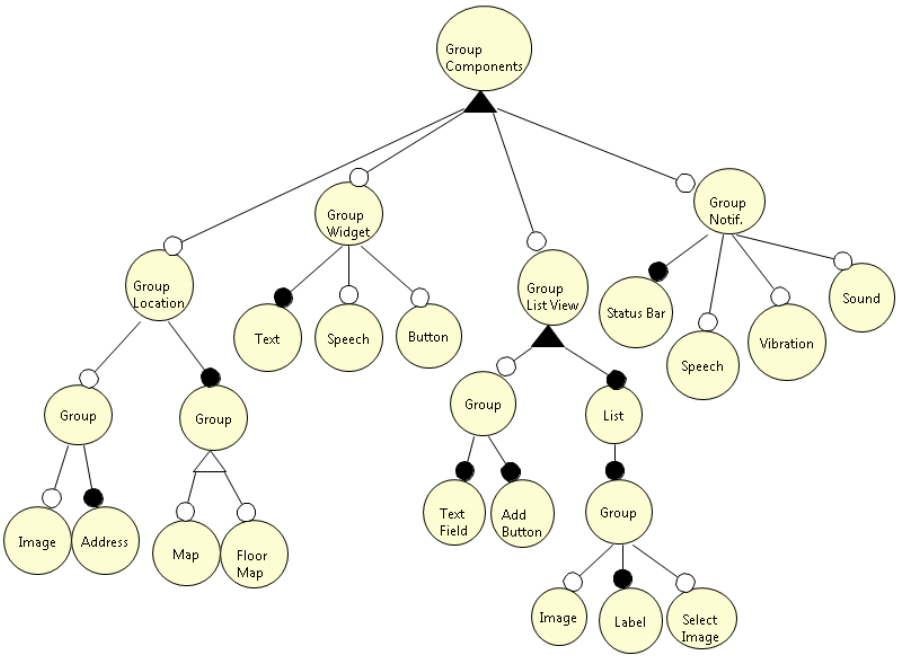

Figure 6.5: Concrete UI components of a Smart Home system.

Linking interaction to components

The concrete UI components that support the different interaction features are specified in Table 6.2

6.1.3 Architecture description

The last step in the design phase is to describe the architecture components of the system. This step is described below.

Interaction feature	Concrete components
Change-based	Group Notif.
Status-based	Group Widget
Graphical	Group Location, Group Notif., Group List View + Image
Text	Text, Address, Group List View
Sound	Sound
Speech	Speech
Vibration	Vibration

Table 6.2: Linking between interaction features and concrete components

7. Specifying the architecture

The final step is to define the way user interfaces of services are integrated with the different components by means of the component architecture model.

Figure 6.6 shows the model for the components of the Smart Home system. We have used a *Service* to represent the functionality of the services defined and *Activities* provide the user interfaces from which service functionality can be accessed. Also, we use a *Service* to launch a notification. A *BroadcastReceiver* is used to provide the functionality of the widget. All the components have defined the intent filters associated to the actions they can perform (e.g. *ADD_ITEM* for the *Add Item* activity). Moreover, the *Show Services* activity has the intent filter *MAIN* to mark this activity as the initial activity. There are some services that launch intents to start an external service (e.g. *Show Location* activity launches the intent *VIEW* to show the map of the location). This is because the functionality required for these services is implemented by an external service. There are three content providers: one for offering the items of the shopping list, another for offering the

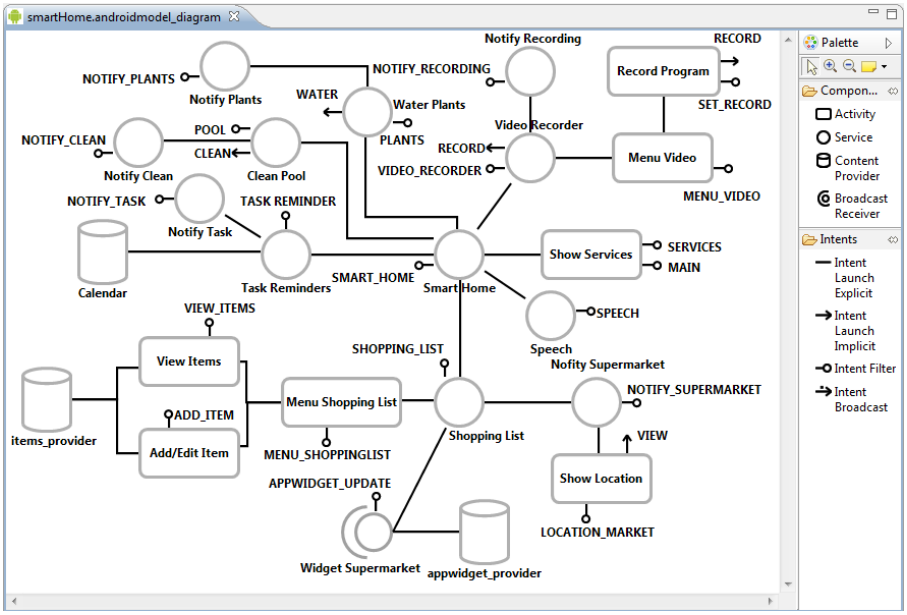

Figure 6.6: Component arquitecture of the Smart Home services.

information to update the *Widget Supermarket* receiver, and another one representing the data of the calendar.

The requirements captured following our method describe the architecture and interaction components to support the Smart Home scenario. Since the design method was followed, the different decisions are organized in different models and consistency is guaranteed among them. Although the models defined are consistent, this does not guarantee that the system described is well accepted by users. Once the system is designed, the following section detects whether the design decisions taken were appropriate.

6.2 Early-stage evaluation

The MDE techniques defined in Chapter 5 can be applied to the previous requirements to obtain a software solution to support the system. However, there is still place for fast-prototyping. In the case of context-aware mobile services, deployment efforts are high if real hardware is used. Designers need certain guaranties that the adaptation defined will fit well when it is finally deployed.

Fast-prototyping techniques have been applied in order to immerse the user in the adaptation designed for mobile services without actually implementing it. Android interface mock-ups and Wizard of Of techniques are used to simulate the process. From the device perspective, it has (1) to be running under Android Operating System and (2) to have wireless connectivity. The physical context was reproduced in a laboratory, setting all the scenarios that appear in the case study.

6.2.1 Questionnaire and participants

Once the user is immersed in the simulated environment, the user experience in terms of usability and interaction adaptation is evaluated. To evaluate these aspects, we used an adapted IBM Post-Study questionnaire (Lewis, 1995) in conjunction with the questionnaire defined by Vastenburg et al. in (Vastenburg et al., 2009). On the one hand, IBM Post-Study is a questionnaire that measure user satisfaction with system usability. On the other hand, some questions were taken from the Vastenburg questionnaire to evaluate home notification systems such as messages acceptability and interaction adaptation. The three dimensions evaluated in our questionnaire were: *Usability of the system*, *messages acceptability* and *interaction adaptation*. The first dimension focuses on measuring users' acceptance with the usability of the system; the second one focuses on the general acceptability considering the messages and the user activity at the time of notification; and finally, the third dimension is about users' satisfaction in the interaction adaptation. Also, we included a NASA task load index (TLX)[1] test. This test

[1] http://humansystems.arc.nasa.gov/groups/TLX/index.html

assesses the user's subjective experience of the overall workload and the factors that contribute to it on six different subscales: *Mental Demand, Physical Demand, Temporal Demand, Performance, Effort*, and *Frustration.*

A total of 15 subjects participated in the experiment (6 female and 9 male). Most of them had a strong background in computer science. Participants were between 23 and 40 years old. 8 out of 15 were familiar with the use of a smartphone, and three own an Android device similar to the one used in the experiment. We applied a Likert scale (from 1 to 5 points) to evaluate the items defined in the questionnaire. Some space was left at the end of the questionnaire for positive and negative aspects, and for further comments.

6.2.2 Procedure

For the evaluation of the Smart Home prototype, users adopted Bob's role and perform the activities earlier described. The study was conducted in our laboratory in order to simulate the different scenarios in which the experiment was based on. In-situ evaluation was possible since the technique does not require a complex infraestructure. An HTC Magic mobile device running Android Operating System was used to interact with the Smart Home services.

When the users evaluated the prototype, they were not told that it was a non-functional prototype. After the evaluation, when they were told that it was not a final functional system, more than a third of the participants confessed that they thought that it was. This means that it is possible to anticipate the feedback that could be obtained from the final system with minimal effort.

6.2.3 Results

Figure 6.7 shows a summarized table of the obtained results[2].

[2]The complete dataset of the experimental results can be downloaded from http://www.pros.upv.es/labs/projects/interactionadaptation

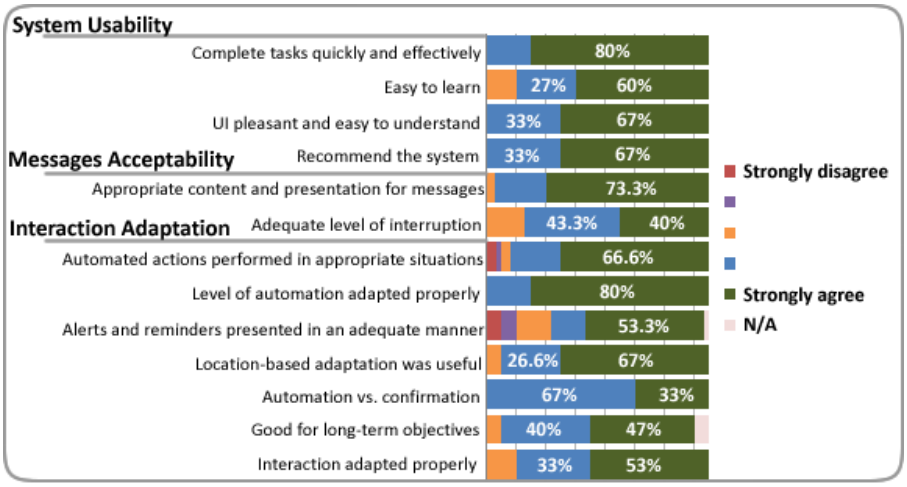

Figure 6.7: Summarized results

More than 70% of the people strongly agreed that using the system they were able to complete the tasks and scenarios effectively and quickly. All users considered (4 or 5 points) the user interface to be pleasant and easy to understand. 67% of users strongly agreed about recommending the system to other people.

With regard to the *messages acceptability*, the results were also positive, but more dispersion was found in them. This was due the different perception each user had about what was considered to be a relevant or urgent message. In the study made by Vastenburg et al. (Vastenburg et al., 2009), they pointed out that the more urgent the message was considered to be, the higher the level of intrusiveness should be. In our results, the content and presentation of the different messages was considered appropiate by the 73% of the subjects. Some users (20%) found some services to be intrusive, but the interruption level was in general (80%) considered adequate to each situation.

Regarding the *interaction adaptation*, automated tasks outcomes are not always discovered (33% of subjects), but 80% of subjects strongly

agreed in that automated actions had performed in appropiate situations and help them to perform routine tasks. There were some exceptions that were suggested in the comments such as "I would like to receive the pool notification and can postpone it" or "When the system clean the pool do not inform the user about that". Although the adaptation provided was considered adequate (more than 80% considered it appropriate for all the services), most of the complaints were related to the level of control provided. Some users would like to be able to undo actions they are notified about such as the video recording, many (67%) did not considered watering the plants deserving a notification, and the suddenly change of the outdoor to an indoor map of the supermarket made some users (33%) feel they were loosing control.

Workload

The results on workload are shown in Figure 6.8. We show each subscale in a different diagram. The *Mental Demand* diagram shows that not all the tasks were simple and easy. Mostly, mental demand was low but some tasks in the experiment required more attention, increasing the mental demand. Some users would prefer more automation in the tasks. *Physical demand* was low excepting for the tasks that require more attention. Moreover, some users were not familiar with the use of a smartphone. For these users, the physical demand was higher.

The low workload was accompanied by good performance. The majority of users could accomplished the goals of the tasks proposed (see the *Performance diagram*) without much effort (see the *Effort diagram*) and with a low degree of frustration (see the *Frustration diagram*). *Temporal demand* did not provide any significant results since the results are very scattered. This results show that users do not understand the question very well.

6.2.4 Adaptation re-design

The feedback from the users was used to re-design the adaptation of the services iteratively. Some concerns resulted in minor modifications

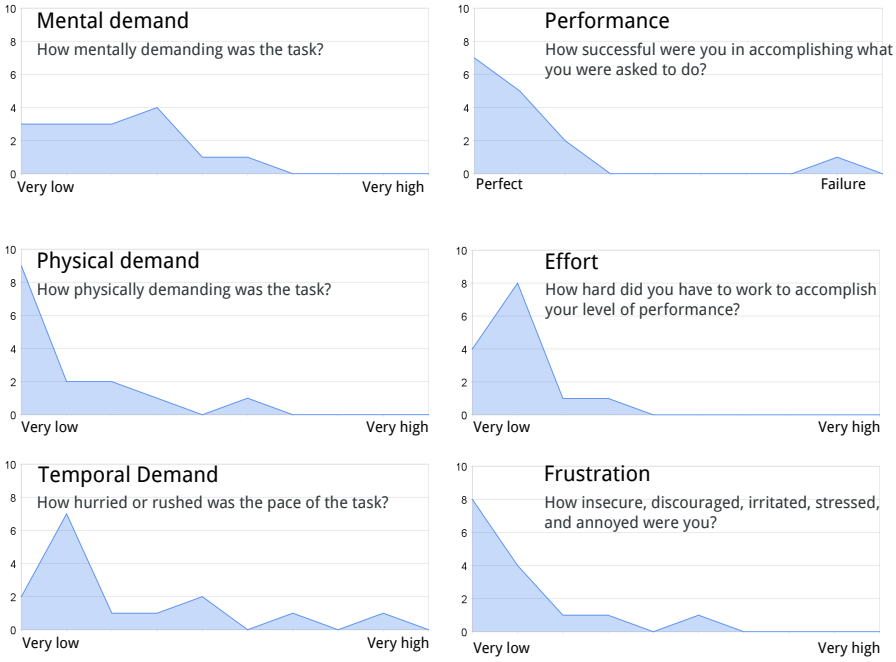

Figure 6.8: Nasa TLX results

of the mock-up interfaces (e.g., using graphical metaphors and bigger buttons). Other suggestions were more relevant to the process redesign since they involved changing the *level of obtrusiveness* for some tasks and providing more contextual information.

For example, the service of *cleaning the pool* was placed in a *proactive-invisible* obtrusiveness level. It means that users was not notified about the cleaning of pool. However, some users wanted to be notified. As a consequence, we changed the obtrusiveness level of clean pool to a *slightly-noticeable* level of attention.

The context conditions associated at *water the plants reminder* was also problematic. In the first iterations of the design, this reminder was in a *slightly-noticeable* level of attention with the condition of using

vibration if users were engaged in other important activities. In the case study, this reminder appeared during the meeting. However, users wanted this reminder only appears when the user was at home.

Users demanded more contextual information for some tasks. For example, in the *notification of supermarket* when the user entered the supermarket, the system showed the floor map of the supermarket, but this information appeared without any other information and users did not know what was happening. To solve this, we added a contextual help.

6.3 Conclusions

Modeling is about abstractions and the conceptualization of the system to be built. However, for a problem to be completely understood analysis hypotheses must be validated with the end-users of the system. In this chapter, we put in practice the design method defined. The application of the development method in the presented case study has provided valuable feedback at different levels. Our research results show that the method is capable of producing services with a high level of user acceptance, and a final software solution can be obtained from what it is captured at requirements level.

The resulting application from this case study are interesting by itself. Although some aspects have been simplified for the development of the prototype, the technologies in use are production ready. In addition, the design effort to improve the case study has lead to better enhancing the user experience and the adaptation fluency.

Chapter 7

Conclusions

The present work has introduced a model-driven development method for developing unobtrusive service interaction. The work focuses on decoupling obtrusivenes, interaction features and UI components in order to minimize efforts in the development. Facing the development of such services from this decomposition has resulted innovative and different contributions were produced from this work. In addition, the research line in which this work is aligned is by no means completed here. Further work can complement and extend this thesis.

This last chapter introduces the conclusions of the work developed in this thesis. First, Section 7.1 presents the main contributions to both the Context-Aware and Considerate communities. Section 7.2 provides an overview of the publications that have emerged from this work. Finally, Section 7.3 outline the ongoing and future work that can extend this research line.

7.1 Contributions

The main contribution of this work is a development process of mobile services that can be adapted in terms of obtrusiveness. The development process comprises from architectural to methodological aspects. So, the work provides the following contributions:

A design method. A design method has been defined allowing to specify services according to the context of each user in terms of obtrusiveness without duplicating efforts in the development.

On the one hand, **we have adapted interaction in terms of attentional demand** accoding to the user needs and the context of use. This has been done by means of the use of (1) personas and (2) the framework for implicit interactions. Thought personas we detect the user needs and contexts of use of each kind of user, define common functionalities and express them in terms of obtrusiveness by means of the framework for implicit interactions.

On the other hand, **we have decomposed interaction in different adaptation aspects** to avoid duplicating efforts in the development. Feature Models allow to decompose interaction aspects and set the constraints for their selection.

A development method. A **systematic development method has been defined** to guide the developer in the construction of unobtrusive mobile services. The method comprises from specification to the final implementation.

As the whole method is supported by models, feedback from users is easily mapped onto the models enabling future improvements. Our approach also allows to visually represent how the interface changes according to different conditions. By analyzing the impact of different factors in the interface we can (1) produce specific variants of the application to target a particular device kind, and (2) define how the interface is adapted at run-time according to different context conditions.

7.2 Publications

Parts of the results presented in this thesis have been presented and discussed before on distinct peer-review forums. The distinct publications in which the author of this thesis was involved are listed below.

1. **Miriam Gil**, Pau Giner & Vicente Pelechano. *Service Obtrusiveness Adaptation.* First International Joint Conference on Ambient Intelligence (AmI), Malaga, Spain, 2010. pp. 11-20. Springer Berlin / Heidelberg. Lecture Notes in Computer Science, Volume 6439.

2. **Miriam Gil**, Pau Giner & Vicente Pelechano. *Designing context-aware mobile interactions.* 4th Symposium of Ubiquitous Computing and Ambient Intelligence (UCAmI), Valencia, Spain, 2010. pp. 93-102.

3. Pablo Munoz, Pau Giner & **Miriam Gil**. *Designing Context-Aware Interactions for Task-Based Applications.* 10th International Conference on Web Engineering, ICWE 2010 Workshops, Vienna, Austria, 2010. pp. 463-473. Springer Berlin / Heidelberg. Lecture Notes in Computer Science, Volume 6385.

4. **Miriam Gil**, Pau Giner & Vicente Pelechano. *Service Obtrusiveness Personalisation.* Pervasive Personalisation Workshop held in conjunction with Pervasive 2010, Helsinki, Finland, 2010. pp. 18-25.

Initial development of the work have been accepted in workshops from relevant conferences such as Pervasive (*Tier-A* according to CORE conference ranking) or ICWE. The modeling approach to adapt user interfaces according to changes in context has been published in the 4th Symposium of Ubiquitous Computing and Ambient Intelligence (UCAmI'10), and the development method of mobile services that can be adapted to regulate the service obtrusiveness has been published in the First International Joint Conference on Ambient Intelligence (AmI'10).

7.2.1 Relevance of the publications

This section provides some information about the relevance of the conferences where different aspects of this work have been published.

Ambient Intelligence. The AmI conference has an important role for the social cohesion of the Ambient Intelligence community. The conference papers are published by Springer (LNCS). The conference has a major impact in industry with the participation of relevant corporations in the AmI area such as Nokia, Philips, NTT DOCOMO or SAP.

Ubiquitous Computing and Ambient Intelligence. UCAmI has been consolidated as a reference event in Ubiquitous Computing & Ambient Intelligence, agglutinating high quality papers.

7.3 Future Work

The research presented here is not a closed work and there are several interesting directions that can be taken to provide the proposal with a wider spectrum of application. The following summarizes the research activities that are planned to continue this work.

End user development. The use of modeling techniques to formalize concepts allows the automation of software development. In the present work, the generation of an initial version of the system that hides infrastructure issues is obtained automatically. The next big step is to provide developers and end-users with tools to intuitively develop their unobtrusive mobile services easily enabling end-users to set their preferences. In order to do so, the primitives that capture the requirements for the system should be provided adequate tool support not only to make feasible the development but also to provide an optimal user experience for developers.

Adaptation visualization. At the moment, our design tool supports different kinds of visualizations to represent the information. We want to extend the tool with a new visualization that highlights the active nodes for a given set of conditions in order to give a more intuitive representation of adaptation.

Run-time reconfiguration. Currently, we are considering the impact of context conditions at design, but we also plan to give support at run-time in order to perform dynamic changes in response to context variations. To this end, we plan to integrate it with the Model-based Reconfiguration Engine (MoRE) (Cetina et al., 2009) to achieve a dynamic reconfiguration in response to context variations.

Bibliography

Aarts, E., Harwig, R., & Schuurmans, M. (2002). Ambient intelligence. *The invisible future: the seamless integration of technology into everyday life*, (pp. 235–250).

Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., & Shuster, J. E. (1999). UIML: An appliance-independent XML user interface language. In *WWW '08: 8th International Conference on World Wide Web*.

Altosaar, M., Vertegaal, R., Sohn, C., & Cheng, D. (2006). Auraorb: social notification appliance. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, (pp. 381–386). New York, NY, USA: ACM.

Ballagas, R., Borchers, J., Rohs, M., & Sheridan, J. G. (2006). The smart phone: A ubiquitous input device. *IEEE Pervasive Computing*, *5*(1), 70.

Balme, L., Demeure, A., Barralon, N., Coutaz, J., & Calvary, G. (2004). Cameleon-rt: A software architecture reference model for distributed, migratable, and plastic user interfaces. In *EUSAI*, (pp. 291–302).

Benavides, D., Cortés, R. A., & Trinidad, P. (2005). Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, *3520*, 491–503.

Bézivin, J., & Gerbé, O. (2001). Towards a precise definition of the OMG/MDA framework. In

ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering, (p. 273). Washington, DC, USA: IEEE Computer Society.

Brown, D. M. (2010). *Communicating Design: Developing Web Site Documentation for Design and Planning (2nd Edition)*. New Riders Press.

Buxton, B. (1995). Integrating the periphery and context: A new model of telematics. In *Proceedings of Graphics Interface*, (pp. 239–246).

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, *15*(3), 289–308.

Carroll, J. M. (2000). *Making Use: Scenario-Based Design of Human-Computer Interactions*. The MIT Press, 1st ed.

Cetina, C., Giner, P., Fons, J., & Pelechano, V. (2009). Autonomic computing through reuse of variability models at runtime: The case of smart homes. *Computer*, *42*(10), 37–43.

Chittaro, L. (2010). Distinctive aspects of mobile interaction and their implications for the design of multimodal interfaces. *Multimodal User Interfaces*, *3*, 157–165.

Clerckx, T., Luyten, K., & Coninx, K. (2004). Dynamo-aid: a design process and a runtime architecture for dynamic model-based user interface development. In *In The 9th IFIP Working Conference on Engineering for Human-Computer Interaction Jointly with The 11th International Workshop on Design, Specification and Verification of Interactive Systems*, (pp. 11–13). Springer-Verlag.

Clerckx, T., Winters, F., & Coninx, K. (2005). Tool support for designing context-sensitive user interfaces using a model-based approach. In *TAMODIA '05: Proceedings of the 4th international workshop on Task models and diagrams*, (pp. 11–18). New York, NY, USA: ACM Press.

Coninx, K., Luyten, K., Vandervelpen, C., V, C., Creemers, B., Bergh, J. V. D., & Centrum, L. U. (2003). Dygimes: Dynamically generating interfaces for mobile computing de-

vices and embedded systems. In *In Human-Computer Interaction with Mobile Devices and Services, 5th International Symposium, Mobile HCI 2003*, (pp. 256–270). Springer.

Cooper, A., Reimann, R., & Cronin, D. (2007). *About Face 3: The Essentials of Interaction Design*. New York, NY, USA: Wiley Publishing, Inc.

Coplien, J., Hoffman, D., & Weiss, D. (1998). Commonality and variability in software engineering. *IEEE Software*, *15*(6), 37–45.

Czarnecki, K., & Kim, P. (2005). Cardinality-based feature modeling and constraints: A progress report. In *Proceedings of the International Workshop on Software Factories At OOPSLA 2005*.

da Silva, P. P., & Paton, N. W. (2003). User interface modeling in UMLi. *IEEE Softw.*, *20*(4), 62–69.

Dahlbäck, N., Jönsson, A., & Ahrenberg, L. (1993). Wizard of Oz studies: why and how. In *IUI '93: Proceedings of the 1st international conference on Intelligent user interfaces*, (pp. 193–200). New York, NY, USA: ACM.

de Sá, M., & Carriço, L. (2006). Low-fi prototyping for mobile devices. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, (pp. 694–699). New York, NY, USA: ACM.

de Sá, M., & Carriço, L. (2009). A mobile tool for in-situ prototyping. In *MobileHCI '09: Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, (pp. 1–4). New York, NY, USA: ACM.

den Bergh, J. V., & Coninx, K. (2005). Towards modeling context-sensitive interactive applications: the context-sensitive user interface profile (CUP). In *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, (pp. 87–94). New York, NY, USA: ACM Press.

Dey, A. K., & Abowd, G. D. (2000). Towards a better understanding of context and context-awareness. In *Computer Hu-*

man Intraction 2000 Workshop on the What, Who, Where,.

DiMarzio, J. (2008). *Android a Programmers Guide*. McGraw-Hill Osborne Media.

Duarte, C., & Carriço, L. (2006). A conceptual framework for developing adaptive multimodal applications. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, (pp. 132–139). New York, NY, USA: ACM.

Dunlop, M., & Brewster, S. (2002). The challenge of mobile devices for human computer interaction. *Personal Ubiquitous Comput.*, *6*(4), 235–236.

Eisenstein, J., Vanderdonckt, J., & Puerta, A. (2001). Applying model-based techniques to the development of uis for mobile computers. In *IUI '01: Proceedings of the 6th international conference on Intelligent user interfaces*, (pp. 69–76). New York, NY, USA: ACM.

Evans (2003). *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Favre, J.-M. (2004). Foundations of Model (Driven) (Reverse) Engineering : Models – Episode I: Stories of the fidus papyrus and of the solarus. In J. Bezivin, & R. Heckel (Eds.) *Language Engineering for Model-Driven Software Development*, no. 04101 in Dagstuhl Seminar Proceedings. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

Fischer, G. (2001). User modeling in human-computer interaction. *User Modeling and User-Adapted Interaction*, *11*(1-2), 65–86.

Gershenfeld, N. (1999). *When Things Start to Think*. New York, NY, USA: Henry Holt and Co., Inc.

Gibbs, W. W. (2004). Considerate computing. *Scientific American*, *292*(1), 54–61.

Giner, P., Cetina, C., Fons, J., & Pelechano, V. (2010). Developing mobile workflow support in the internet of things. *IEEE Pervasive Computing*, *9*(2), 18–26.

Greenfield, A. (2006). *Everyware: The Dawning Age of Ubiquitous Computing*. Berkeley, CA: New Riders Publishing.

Gulliksen, J., Goransson, B., Boivie, I., Blomkvist, S., Persson, J., & Cajander, A. (2003). Key principles for user-centred systems design. *Behaviour & Information Technolog*, *22*, 397 409.

Hagen, P., Robertson, T., Kan, M., & Sadler, K. (2005). Emerging research methods for understanding mobile technology use. In *OZCHI '05: Proceedings of the 17th Australia conference on Computer-Human Interaction*, (pp. 1–10). Narrabundah, Australia, Australia: Computer-Human Interaction Special Interest Group (CHISIG) of Australia.

Hansmann, U., Nicklous, M. S., & Stober, T. (2001). *Pervasive computing handbook*. New York, NY, USA: Springer-Verlag New York, Inc.

Henricksen, K., & Indulska, J. (2005). Developing context-aware pervasive computing applications: models and approach. In *Pervasive and Mobile Computing, In*. Press, Elsevier.

Hinckley, K., & Horvitz, E. (2001). Toward more sensitive mobile phones. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, (pp. 191–192). New York, NY, USA: ACM.

Ho, J., & Intille, S. S. (2005). Using context-aware computing to reduce the perceived burden of interruptions from mobile devices. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 909–918). New York, NY, USA: ACM.

Horvitz, E., Kadie, C., Paek, T., & Hovel, D. (2003). Models of attention in computing and communication: from principles to applications. *Commun. ACM*, *46*(3), 52–59.

Horvitz, E., Koch, P., & Apacible, J. (2004). Busybody: creating and fielding personalized models of the cost of interruption. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative*

work, (pp. 507–510). New York, NY, USA: ACM.

Huberman, B. A., & Wu, F. (2007). The economics of attention: maximizing user value in information-rich environments. In *ADKDD '07: Proceedings of the 1st international workshop on Data mining and audience intelligence for advertising*, (pp. 16–20). New York, NY, USA: ACM.

Jan Willem Streefkerk, M. P. v. E.-B., & Neerincx, M. A. (2006). Designing personal attentive user interfaces in the mobile public safety domain. *Computers in Human Behavior*, *22*, 749–770.

Ju, W., & Leifer, L. (2008). The design of implicit interactions: Making interactive systems less obnoxious. *Design Issues*, *24*(3), 72–84.

Kramer, J. (2007). Is abstraction the key to computing? *Commun. ACM*, *50*(4), 36–42.

Krogstie, J., Lyytinen, K., Opdahl, A. L., Pernici, B., Siau, K., & Smolander, K. (2004). Research areas and challenges for mobile information systems. *Int. J. Mob. Commun.*, *2*(3), 220–234.

Lemmelä, S., Vetek, A., Mäkelä, K., & Trendafilov, D. (2008). Designing and evaluating multimodal interaction for mobile contexts. In *IMCI '08: Proceedings of the 10th international conference on Multimodal interfaces*, (pp. 265–272). New York, NY, USA: ACM.

Lewis, J. R. (1995). Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *Int. J. Hum.-Comput. Interact.*, *7*(1), 57–78.

Limbourg, Q., V, J., Michotte, B., Bouillon, L., Florins, M., & Trevisan, D. (2004). Usixml: A user interface description language for context-sensitive user interfaces. In *in Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages*, (pp. 55–62). Press.

Lucas, P. (2001). Mobile devices and mobile data: issues of identity and refence. *Hum.-Comput. Interact.*, *16*(2), 323–336.

Maeda, J. (2006). *The Laws of Simplicity (Simplicity: Design, Technology, Business, Life)*. The MIT Press.

Maiden, N. (2009). Where are we? handling context. *IEEE Software*, *26*(5), 75–76.

Mao, J.-Y., Vredenburg, K., Smith, P. W., & Carey, T. (2001). User-centered design methods in practice: a survey of the state of the art. In *CASCON '01: Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research*, (p. 12). IBM Press.

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decis. Support Syst.*, *15*(4), 251–266.

Miller, J., & Mukerji, J. (2003). MDA Guide Version 1.0.1. Tech. rep., Object Management Group (OMG).

Mori, G., Paternò, F., & Santoro, C. (2002). Ctte: Support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering*, *28*(8), 797–813.

Mori, G., Paternò, F., & Santoro, C. (2004). Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Trans. Softw. Eng.*, *30*(8), 507–520.

Neely, S., Stevenson, G., Kray, C., Mulder, I., Connelly, K., & Siek, K. A. (2008). Evaluating pervasive and ubiquitous systems. *IEEE Pervasive Computing*, *7*(3), 85–88.

O'Grady, M., O'Hare, G., & Keegan, S. (2008). Interaction modalities in mobile contexts. In M. Virvou, & L. Jain (Eds.) *Intelligent Interactive Systems in Knowledge-Based Environments*, vol. 104 of *Studies in Computational Intelligence*, (pp. 89–106). Springer Berlin / Heidelberg. 10.1007/978-3-540-77471-6_6.

OMG (2006). Business Process Modeling Notation (BPMN) Specification. OMG Final Adopted Specification. dtc/06-02-01.

Pederiva, I., Vanderdonckt, J., España, S., Panach, J. I., & Pastor, O. (2007). The beautification process in model-driven engineering of user interfaces. In *11th IFIP TC 13 Int. Conf. on Human-Computer*

Interaction INTERACT2007, (pp. 411–425).

Puerta, A. (1996). The mecano project: Comprehensive and integrated support for model-based interface development. In *In Computer-Aided Design of User Interfaces*, (pp. 5–7). Namur University Press.

Puerta, A., & Eisenstein, J. (2002). Ximl: a common representation for interaction data. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, (pp. 214–215). New York, NY, USA: ACM.

Ramchurn, S. D., Deitch, B., Thompson, M. K., Roure, D. C. D., Jennings, N. R., & Luck, M. (2004). Minimising intrusiveness in pervasive computing environments using multi-agent negotiation. *Mobile and Ubiquitous Systems, Annual International Conference on*, *0*, 364–372.

Rumbaugh, J., Jacobson, I., & Booch, G. (1998). *The Unified Modeling Language Reference Manual*. Addison-Wesley.

Satyanarayanan, M. (1996). Fundamental challenges in mobile computing. In *PODC '96: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, (pp. 1–7). New York, NY, USA: ACM.

Schmidt, A. (2000). Implicit human computer interaction through context. Tech. rep., Personal Technologies.

Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *Computer*, *39*(2), 25–31.

Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., & Bontemps, Y. (2007). Generic semantics of feature diagrams. *Comput. Netw.*, *51*(2), 456–479.

Seffah, A., Forbrig, P., & Javahery, H. (2004). Multi-devices "multiple" user interfaces: development models and research opportunities. *J. Syst. Softw.*, *73*(2), 287–300.

Sharp, H., Rogers, Y., & Preece, J. (2007). *Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons.

Siau, K. (2003). *Advances in Mobile Commerce Technologies*. Hershey, PA, USA: IGI Publishing.

Siewiorek, D., Smailagic, A., Furukawa, J., Krause, A., Moraveji,

N., Reiger, K., Shaffer, J., & Wong, F. L. (2003). Sensay: A context-aware mobile phone. In *ISWC '03: Proceedings of the 7th IEEE International Symposium on Wearable Computers*, (p. 248). Washington, DC, USA: IEEE Computer Society.

Silva, P. P. D. (2000). User interface declarative models and development environments: A survey. In *Proceedings of DSV-IS2000, volume 1946 of LNCS*, (pp. 207–226). Springer-Verlag.

Sottet, J.-S., Calvary, G., & Favre, J.-M. (2005). Towards model-driven engineering of plastic user interfaces. In *MDDAUI*.

Tamminen, S., Oulasvirta, A., Toiskallio, K., & Kankainen, A. (2004). Understanding mobile contexts. *Personal Ubiquitous Comput.*, *8*(2), 135–143.

Tedre, M. (2008). What should be automated? *interactions*, *15*(5), 47–49.

Thevenin, D., & Coutaz, J. (1999). Plasticity of user interfaces: Framework and research agenda. In A. Sasse, & C. Johnson (Eds.) *Proceedings of IFIP Conference on Human-Computer Interaction Interact'99*, (pp. 110–117). IOS Press Publ.

Vaishnavi, V., & Kuechler, W. (2004). Design research in information systems. http://desrist.org/design-research-in-information-systems.

Van den Bergh, J., & Coninx, K. (2004). Model-based design of context-sensitive interactive applications: a discussion of notations. In *TAMODIA '04: Proceedings of the 3rd annual conference on Task models and diagrams*, (pp. 43–50). New York, NY, USA: ACM.

Van den Bergh, J., & Coninx, K. (2005). Using uml 2.0 and profiles for modelling context-sensitive user interfaces. In *Proceedings of the MDDAUI2005 CEUR Workshop*. CEUR.

van Deursen, A., Klint, P., & Visser, J. (2000). Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, *35*(6), 26–36.

Vastenburg, M. H., Keyson, D. V., & de Ridder, H. (2008). Considerate home notification systems: a field study of acceptability of notifications in the home. *Personal and Ubiquitous Computing*, *12*(8), 555–566.

Vastenburg, M. H., Keyson, D. V., & de Ridder, H. (2009). Considerate home notification systems: A user study of acceptability of notifications in a living-room laboratory. *Int. J. Hum.-Comput. Stud.*, *67*(9), 814–826.

Vertegaal, R. (2003). Attentive user interfaces. *Commun. ACM*, *46*(3), 30–33.

Vertegaal, R., Shell, J. S., Chen, D., & Mamuji, A. (2006). Designing for augmented attention: Towards a framework for attentive user interfaces. *Computers in Human Behavior*, *22*(4), 771–789.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, *265*(3), 66–75.

Weiser, M., & Brown, J. S. (1997). The coming age of calm technolgy. (pp. 75–85).

Yamabe, T., & Takahashi, K. (2007). Experiments in mobile user interface adaptation for walking users. In *IPC '07: Proceedings of the The 2007 International Conference on Intelligent Pervasive Computing*, (pp. 280–284). Washington, DC, USA: IEEE Computer Society.

www.pros.upv.es

Centro de Investigación en Métodos
de Producción de Software
Universidad Politécnica de Valencia
Camino de Vera s/n
Building 1F
46007 Valencia
Spain
Tel: (+34) 963 877 007 (Ext. 83530)
Fax: (+34) 963 877 359

UNIVERSIDAD
POLITECNICA
DE VALENCIA