Gil Pascual, M.; Giner Blasco, P.; Pelechano Ferragud, V. (2012). Personalization for unobtrusive service interaction. Personal and Ubiquitous Computing. 16(5):543-561. doi:10.1007/s00779-011-0414-0.

Pers Ubiquit Comp manuscript No.
(will be inserted by the editor)

Personalization for unobtrusive service interaction

Miriam Gil · Pau Giner · Vicente Pelechano

Received: date / Accepted: date

Abstract Increasingly, mobile devices play a key role in the communication between users and the services embedded in their environment. With ever greater number of services added to our surroundings, there is a need to personalize services according to the user needs and environmental context avoiding service behavior from becoming overwhelming. In order to prevent this information overload, we present a method for the development of mobile services that can be personalized in terms of obtrusiveness (the degree in which each service intrudes the user's mind) according to the user needs and preferences. That is, services can be developed to provide their functionality at different obtrusiveness levels depending on the user by minimizing the duplication of efforts. On the one hand, we provide mechanisms for describing the obtrusiveness degree required for a service. On the other hand, we make use of Feature Modeling techniques in order to define the obtrusiveness level adaptation in a declarative manner. An experiment was conducted in order to put in practice the proposal and evaluate the user acceptance for the personalization capabilities provided by our approach.

M. Gil · P. Giner · V. Pelechano
Centro de Investigación en Métodos de Producción de Software,
Universitat Politècnica de València,
Camino de Vera s/n, 46022 Valencia, Spain
Tel.: +34-963877007 (Ext. 73551), Fax: +34-963877359
E-mail: mgil@pros.upv.es

P. Giner
pginer@pros.upv.es

V. Pelechano
pele@pros.upv.es

Keywords Obtrusiveness adaptation · Personalization · Feature modeling · Interaction adaptation · Context-awareness

1 Introduction

In ubiquitous computing environments, 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 [45]. The advanced capabilities of mobile devices give them great potential to be the default physical interface for ubiquitous computing [2]. 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 are not disturbing for them. Since user attention is a valuable but limited resource, an environment full of embedded services must behave in a considerate manner [18], demanding user attention only when it is actually required. Evaluating Presto [19], a context aware mobile platform that allows to support different workflows by interacting with the physical environment, we found the need for mechanisms that adapt the degree to which interaction intrudes on user attention.

To avoid service behavior from becoming overwhelming, we propose a technique to adjust the way attentional resources of each user are considered by pervasive services according to the user needs.

Since mobile devices provide a rich contextual information about the user, the system can anticipate some of the user tasks in order not to interrupt him/her. However a complete automation is not always possible or desirable [41]. For certain tasks, some users prefer that the system acts silently in order not to be dis-

turbed. 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 their preferences. But, 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.).

The main contribution of this work is a method for the development of mobile services that can be **personalized** to regulate the service obtrusiveness (i.e., the extent to which each service intrudes the user's mind) in a systematic way. In this work, we enrich the design phase with information (abstractions) to define the attentional resources used for each service depending on the user needs. In this way, user needs drive the design of the system providing users with personalized services and avoiding information overload.

Our approach relies on proven frameworks and modeling techniques. On the one hand, personas [8] are user profiles used to gather the relevant information of the audience helping to drive design and to detect common functionalities between users. In order to provide personalized services according to the user needs, these needs have to be known by means of user modeling techniques. Moreover, we have applied this personalization technique to express the user needs in terms of obtrusiveness. On the other hand, Feature Modeling techniques [13] are applied to describe the commonalities and differences between the interaction mechanisms provided for each service and the constraints for their selection. We have defined user interfaces by means of UI fragments. In this way, Feature Models allow designers to represent these fragments in an abstract manner in a way that a feature can be implemented by one or many UI fragments. Thus, Feature Models are used to define the different alternatives for combining these fragments according to the service obtrusiveness required for each user.

This paper is organized as follows. Section 2 describes the development method defined for supporting our approach. Section 3 provides detail on the tool support developed. Section 4 describes the application of the proposal in a case study that is based on a Smart Home scenario and presents the results from this application. Section 5 presents related work. In Section 6, a discussion of the usefulness and efficiency of the proposed method is introduced. Finally, Section 7 concludes the paper.

2 Developing personalized mobile services

The goal of the approach presented in this work is to manage the attentional demand of services according to user needs in order to avoid information overload.

We propose a technique to personalize the way in which a pervasive service is accessed by adjusting its obtrusiveness level. For defining such services we need to (1) detect the user needs and preferences to determine the obtrusiveness level required for the interaction and (2) make use of the adequate interaction mechanisms to provide the functionality according to this obtrusiveness level.

To understand the users and capture their needs and preferences we use *personas*. 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 [37], ConcurTaskTrees [30], and Business Process Modeling Notation (BPMN) [33]. 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 [21]. Thus, it is interesting the use of them in this work where we have directly address specific user needs following a user-centered design.

For the selection of the adequate interaction mechanism we make use of Feature Models. Feature Models allow to describe the essential aspects of each interaction mechanism and the ways in which they can be combined. 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.

This work is based on Model Driven Engineering (MDE) principles [16] in order to automate the development of personalized mobile services in a systematic way. 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 abstract descriptions. Following the development process iteratively, a prototype of the system is obtained. Then, following a user-centered design, feedback from users drives changes in requirements and the detection of new adaptation aspects. In this way, the initial prototype can be evolved to the final solution.

Thus, we have defined a method to guide the development of personalized services based on models and descriptions. The method comprises three phases: *analysis*, *creation*, and *interface generation*. The development process is shown in Figure 1. The different coor-

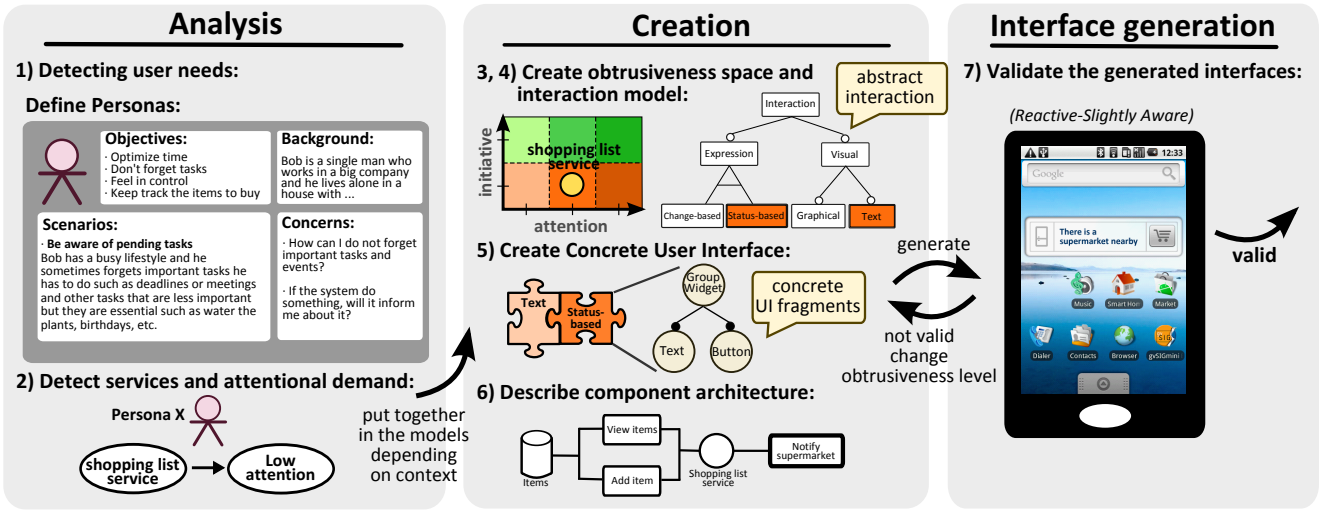

Fig. 1 Development process

dinated activities that constitute the development process, the roles and the models involved are detailed below.

2.1 Analysis phase

User modeling becomes the first stage in the development of a new personalized system in order to capture user requirements. Following the user-centered design principles [28], this phase requires several iterations to ensure that the captured user requirements fulfill the real needs. There is a team of *designers* taking part in different interaction design roles (information architect, interaction designer, and user researcher) [42] in charge of this phase. The activities involved in this stage are the following:

2.1.1 Detecting user needs

The first step in the personalization of pervasive services is to understand who the users will be by studying their cognitive, behavioral and attitudinal characteristics. In order to do this we make use of **Personas** (also known as User Profiles). A persona is a summary representation of the system's intended users, often described as real people [8]. They provide a framework for describing the target audience in a way useful to design and personalize systems.

Personas (or User Profiles) describe target users of the system, giving a clear picture of how they are likely to use the system, and what they will expect from it [8]. Personas capture relevant information about users that directly impact on the design process: user goals, scenarios, tasks, and the like. Although these user profiles are depicted as specific individuals because they function as archetypes, they represent a type of user. Users are grouped into personas, and the personas are analyzed to facilitate service personalization at a person level.

However, a user does not always need to be of the same type. A user can evolve and services have to be continuosly adapted to the needs of each moment. For example, in an online banking system, the needs of a user can evolve from the *New Customer* group to the *Regular User* group. So, the system will have to adapt the services provided based on the new profile.

Personas are synthesized from data collected from *user research* or information-gathering methods such as interviews with users, user testing, etc. In the predesign phase the design team makes interviews and observations that are the basis for creating personas [6]. Then, in this phase, designers analyze the information collected and define the personas.

There is no standard format for personas, and different approaches are offered. Regardless of the selected approach, personas should express what users need and what they expect, containing the majority of the user research findings. In this work, we follow the notation defined in [8] to determine the needs of each user and the functionality required.

In this notation, the information is structured following three layers of detail. Table 1 shows the elements of a persona prioritized into these 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, *descriptive dimensions* that are individual scales representing knowledge, tasks, interests and characteristics, the *objectives and motivations* of the

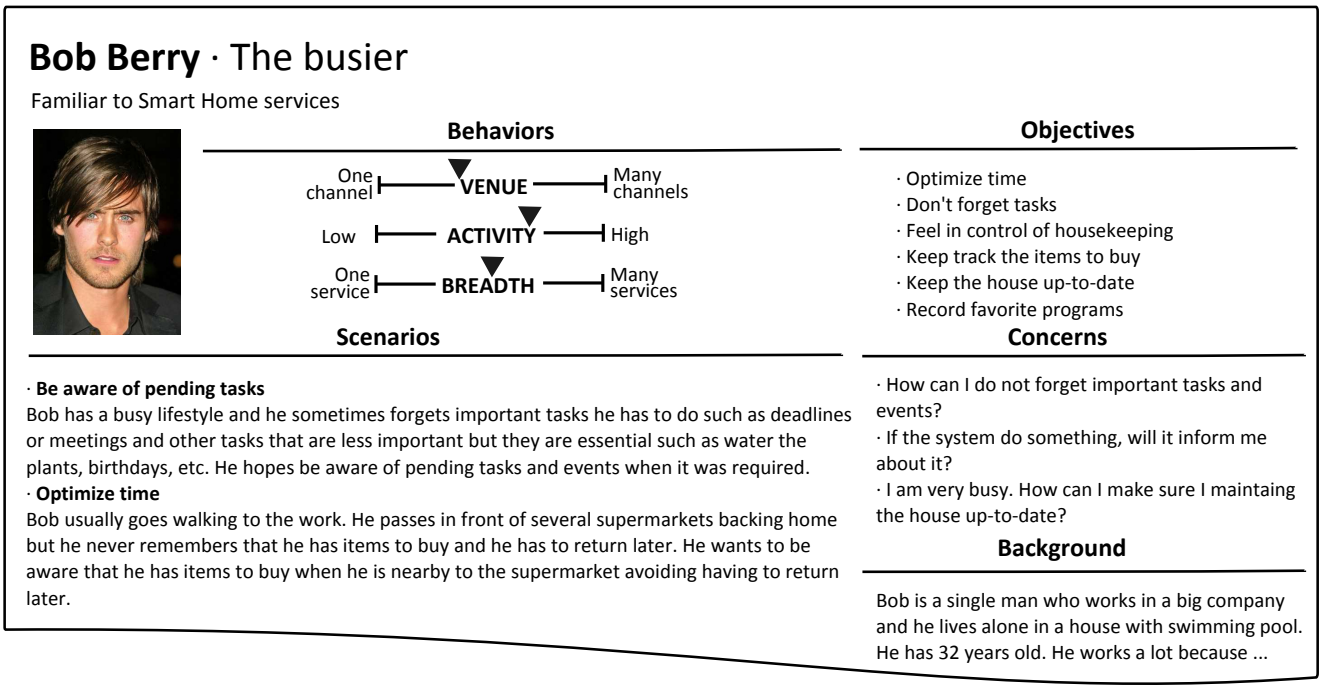

Fig. 2 Excerpt of a persona

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

Table 1 The elements of a persona prioritized into three layers [8]

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*, a *photograph* of the persona, etc (see Table 1).

According to these elements that characterize a persona, designers can define the functionalities and tasks that user needs to achieve their objectives and motivations. Moreover, it can be detected common functionalities between personas and these functionalities be expressed in terms of obtrusiveness. Considering system services in the context of a type of user makes easy to determine the way to provide a service personalized to the user needs. For example, services for a *busy user* have to be defined avoiding overwhelming user attention.

Figure 2 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 to simplify his life and to help him in optimizing his time. This excerpt of a persona provides the basics of a user's needs and behaviors. Through careful analysis of this persona, designers can deduce that (1) the user wants Smart Home services for helping him in home tasks to do not waste time, (2) he wants be aware of pending tasks related to home and work, (3) he hopes to be alerted of the updates that services perform, and (4) he prefers as many services as possible.

Information captured in the personas corresponds to the user requirements or needs structured in goals-scenarios-system features. Designers use this understanding of people to determine *what* services personas require to accomplish their goals and *how* the services are presented in terms of obtrusiveness. This is done by the designer manually since there is no explicit characteristic of the impact of obtrusiveness on the requirements. Then, this information will be formalized in the models of the next phase by the designer in order to be processed automatically in the development phases. The way in which services and obtrusiveness are detected is thoroughly described in the next section.

The purpose of personas is not to give a complete theoretical model of a user. Instead, it is aiming at a simple, but good enough description of the user to allow designers to detect the services needed and the level of obtrusiveness which need each type of user.

2.1.2 Detecting services and obtrusiveness

From the definition of personas, designers have to determine **what** information and capabilities our personas require to achieve their needs and **how** this information is provided in terms of obtrusiveness. This is performed by detecting the services of the system and their obtrusiveness degree according to the user context (see step 2 of Figure 1). By establishing the degree of user attention that a task need, we avoid developing overwhelming services. These concepts are expressed together in the models of the next phase.

For example, the services detected from the synthesis of the persona of Figure 2 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.

For these services detected, the degree of attentional demand required according to the user context is: *low attention* for managing the shopping list, *slightly attention* for the video recorder service and *high attention* for the supermarket notification. The agenda to notify important tasks could require *slightly* or *high attention* depending on the priority of the task for the user. This priority can be set up by the user in their preferences.

Some other personas could require other services and different attentional demand for information presentation and interaction with the services depending on their needs. Thus, personas will guide subsequent adaptations in information presentation, modality and interaction style. In order to personalize the services and provide them in a degree of obtrusiveness that fits into each user type, designers define the services in terms of obtrusiveness in the *creation phase* according to the personas.

2.2 Creation phase

Once the user requirements are captured, the different models that characterize personalization and interaction are defined and the mappings among these models are specified in this phase. First, the attentional demand required for each service is defined in terms of obtrusiveness according to the personas analysis. Once the obtrusiveness level for each service is specified, the appropriate interaction technique can be selected from the ones available. These abstract models are complemented with others that provide a more concrete representation of the service components. These concrete models are (1) the concrete interaction components that are going to represent the user interface elements available, and (2) the architecture of the components that form the system. The *interaction designer* is the role in charge of this phase (henceforth we will refer to it as designers).

The different steps carried out in this phase and the models involved are detailed below:

2.2.1 Adjusting the obtrusiveness level

We make use of the conceptual framework presented in [25] to determine the **obtrusiveness level** for each interaction in the system. This framework defines two dimensions 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 interaction with the system is unadvised).

In this work we assume that the different services detected from the user profiles can be situated in a different position of the obtrusiveness space according to the attentional demand required for each type of user.

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 a 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.

Figure 3 illustrates an example of different services in the obtrusiveness space for two personas. In order to highlight the similarities and differences of the needs and tasks of each persona in terms of obtrusiveness we can create a simple table of needs comparison, using circles with shaded pie pieces to indicate the priorities (see the figure). This table indicates the relative level or importance of each task for each persona.

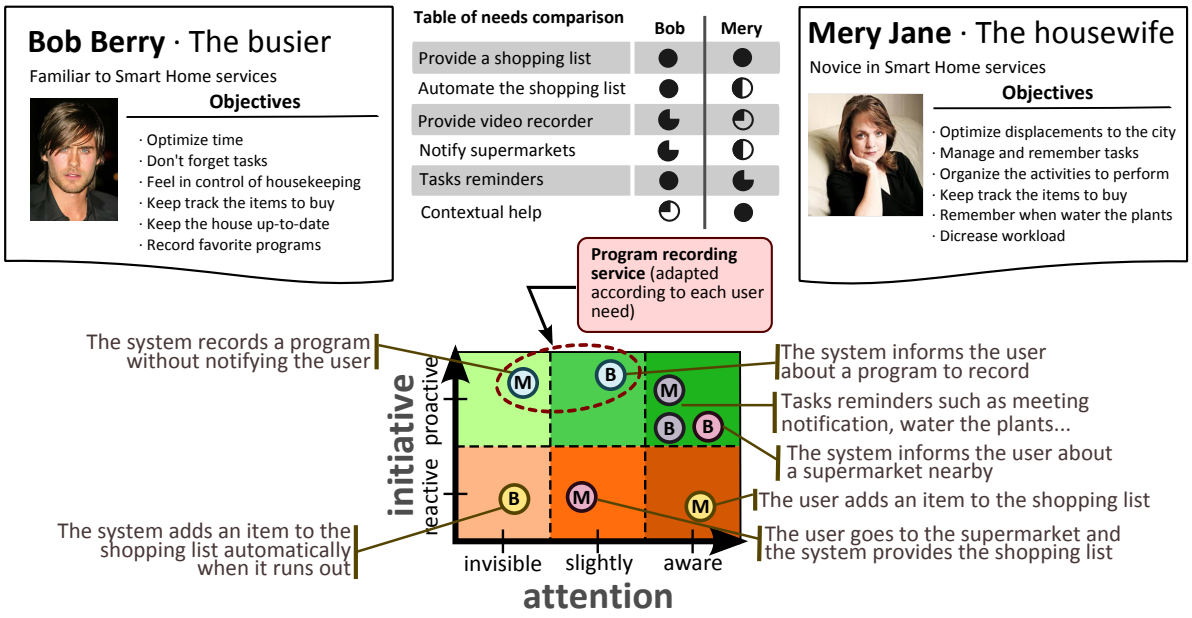

Fig. 3 Services at different obtrusiveness level according to personas

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 our approach we use these divisions to drive the selection of the interaction mechanism that is better suited for each persona.

In the example of the Figure 3, we can see that the same service 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 the programs automatically as it was captured by the persona model (see Figure 2). 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 user needs and priorities. For example, the obtrusiveness level for the notification of a supermarket nearby can be changed depending on the user's location, but can also be changed depending on the priority it has for the user (e.g., demanding more attention when the supermarket is closer or when the items to buy exceed a fixed number). In addition, a particular user X can play Bob and Mary roles at different moment (e.g., weekdays vs. weekends), and the system will provide their services at different obtrusiveness levels according to it. This evolution in the obtrusiveness level is further described in the continuous evolution subsection.

2.2.2 Decomposing interaction aspects

To make use of the interaction mechanism that supports the adequate obtrusiveness level, this work proposes decomposing the context conditions (adaptation aspects) in their *features* (capabilities and limitations). These features are used to describe the interaction in an abstract manner. Feature Modeling is a technique to specify the variants of a system in terms of features (coarse-grained system functionality). 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-

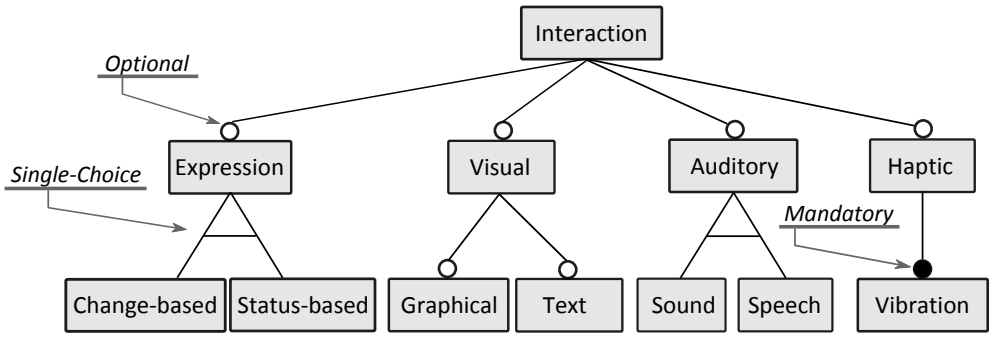

Fig. 4 Feature Model of interaction mechanisms

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.

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 of the system, Feature Models have proven to be effective in hiding much of the complexity in the definition of the adaptation space [10]. 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 Feature Model showed in Figure 4 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).

Feature Models allow us to decompose the interaction in different adaptation aspects without explicitly having to define it for each possible combination of context conditions. This avoids duplicating efforts in the development.

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 designers have about interaction. In the example, we have considered that an interaction element can either be visual or auditory, which is obviously a simplification

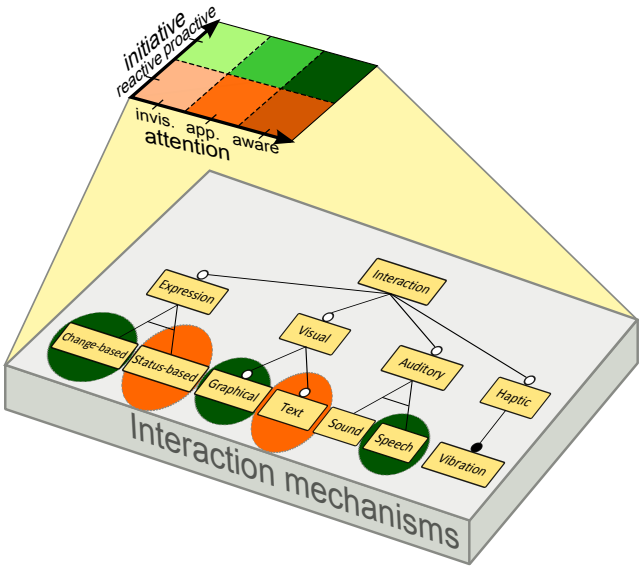

Fig. 5 Mappings between obtrusiveness aspects and interaction mechanisms

since many common widgets normally combine these aspects (e.g., to offer feedback to the user).

2.2.3 Mapping to interaction features

Designers must define the appropriate 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. This constitutes a configuration of interaction for a given obtrusiveness level. Note that the set of the selected interaction features must fulfill the constraints among them represented by their relationships. In order to determine the fulfillment of the constraints among the subset of features, analysis tools such as FAMA [4] can be used.

Figure 5 shows an example of the mapping between obtrusiveness levels and interaction techniques. For ex-

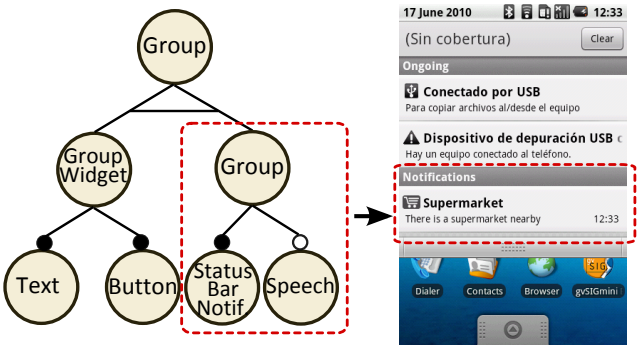

Fig. 6 Concrete Interface model of the "Supermarket Notification"

ample, 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 obtrusiveness aspect).

2.2.4 Concrete interaction 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 interaction components we assume a user interface model that is organized in a *tree structure* allowing a flexible composition of the interaction elements. 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, multi-touch interactions and visual effects as seen in iPhone or Android UIs. This node-based user interface 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 components that the user can perceive such as graphical objects, text, image viewers, UI controls, video viewers, etc.

An example of the concrete user interface model is shown in Figure 6. This example shows the user interface components for a supermarket notification. Depending on the user needs and preferences described in the persona model, the notification would be shown by either a *widget* (left branch) or a *status bar notification* (right branch). For the Bob's persona, a *status bar notification* is chosen since Bob prefers to be completely aware of this kind of notifications (see Figure 3). *Speech* component could be used together to provide a speech interaction (it has an optional constraint) depending on the context conditions (e.g., if the user is alone and he/she is not in a noisy environment). The final user

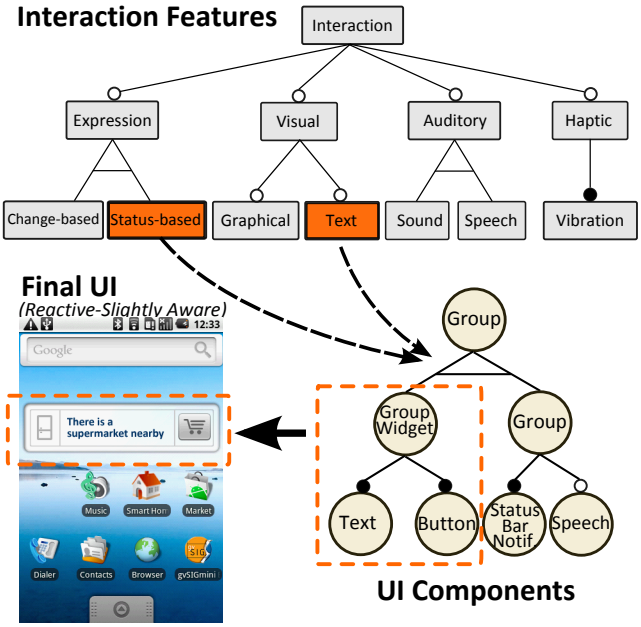

Fig. 7 Mappings between interaction features and concrete components

interface corresponding to this notification is shown at the right of the figure. However, for Mery's persona a widget is preferred according to their needs (subtle interaction) activating the left branch 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 interaction features activated.

2.2.5 Mapping to the concrete interface

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 is activated for a given service, the corresponding concrete UI components are activated too obtaining a personalized user interface.

Figure 7 shows an example of 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

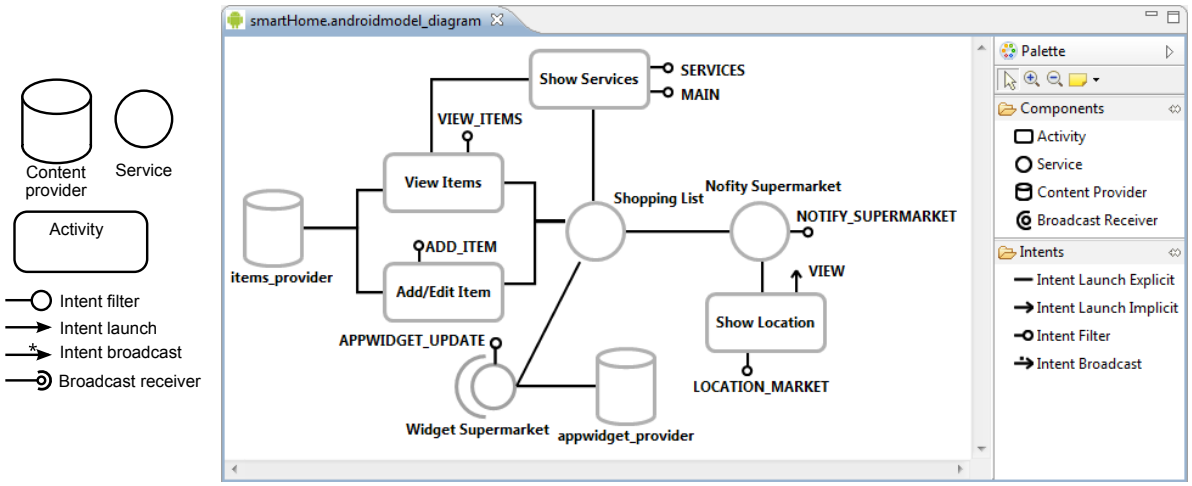

Fig. 8 Component Architecture Model

activated. In particular, the concrete components that support these interaction features are the *Group Widget* node (to support the *status-based* interaction) that contains a *Text* node to show the information and a *Button* to obtain a detailed information.

These mappings are specified by the designer once after the design of the involved models. For later adaptations, these mappings are already defined and interaction is adapted and personalized automatically according to them.

2.2.6 Describing the architecture

The component architecture defines the components that form a given application and the communication that is produced among them. This allows designers to express the dependencies of an application in terms of data and functionality, and detect the sources of context information that can trigger a user interface adaptation. This model is also used to generate the system architecture automatically.

The mobile services developed are based on the Android platform[1]. We have chosen a specific platform because we want (1) to address UI definition at a concrete level of abstraction and (2) to use concepts that are easy to project to the implementation of the concrete platform. In this way, the rules imposed by the platform are respected without dealing with technical implementation details.

The Android platform provides loosely-coupled components such as *Service*, *Activity*, *Content Provider*, and *Broadcast Receiver*. A *Service* in Android provides functionality that is executed in the background, and an *Activity* provides the user interface from which service functionality can be accessed. A *Content Provider* offers data to other components, and a *Broadcast Receiver* is a component that reacts to announcements from other components. Broadcasts are useful to support reactive behavior. The communication mechanisms defined among Android components are 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.

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 *Content Provider (Repository)* used in Android is the same than the described by the Domain Driven Design [15]. Additionally, another specific components such as *intents* are included to support a mobile architecture.

In our system, we use *Services* to represent the functionality of the defined services and *Activities* to provide the user interfaces from which service functionality can be accessed.

Figure 8 shows the model for the components of a shopping list and supermarket notification services. The notation used is illustrated at the left of the figure. 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

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

in the system: the *Shopping List* service in charge of orchestrating the communication between 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. This is because 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 is worth noting that 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.

2.3 Interface generation phase

Once the models are defined and the mappings between the models are specified, the final code for the interface can be generated from the concrete UI model and the architecture components model. A prototype of the final interface can be obtained from this *interface generation phase* to be validated. This phase can be completely automated by means of model transformation techniques. *System developers* are responsible of this phase. Although this is an automated phase, system developers are in charge of creating the generation templates. It is supported by the following step:

2.3.1 UI generation

In this step, the final user interface is made up by those fragments of user interface whose nodes are activated from the node-based structure. The user interface model organized in a tree structure used provides a flexible composition of the user interface elements and an easier node substitution. In Figure 7 we can see the user interface generated for the active nodes.

The implementation code of the system is generated by means of model-to-code transformations. These transformations are implemented using XPand templates from the Model-to-Text (M2T) project[2], 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.

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

The current implementation provides code-generation capabilities for two different aspects: (1) the architecture components of the whole system, and (2) the different user interfaces.

On the one hand, we provide generation for the Android components of the whole application defined from the component architecture model. This includes the generation of the *Android Manifest* and the different *Android classes* that are required for the implementation of the different components. 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.

On the other hand, we generate the user interfaces from the concrete UI model. In particular, in an Android application, the user interface is defined using a hierarchy of *View* and *ViewGroup* nodes. 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. So, we provide generation capabilities to generate the Android *XML layout file* for the *Android Activity* classes that correspond to the different user interfaces.

The advantage of declaring the UI in XML is that it facilitates to 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 cannot be implemented by means of the layout file. The status bar notification is initiated from a *Service*. In this way, the notification can be created

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

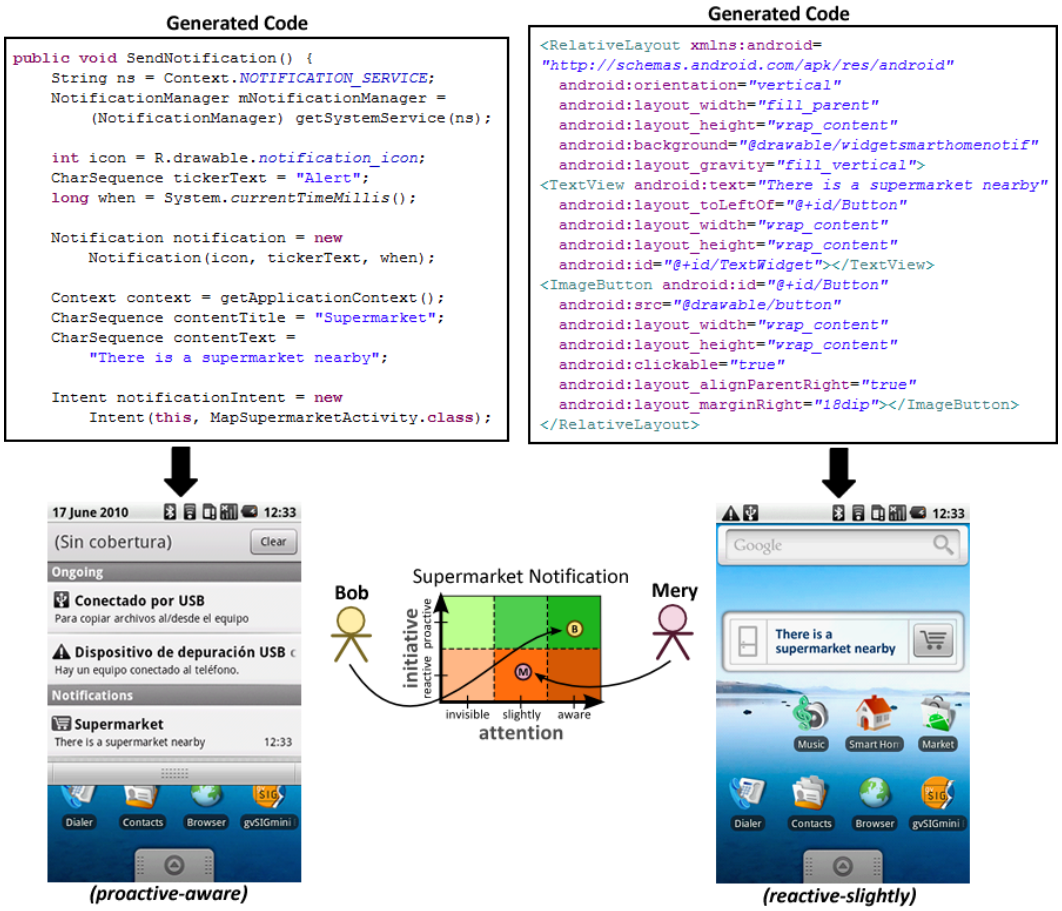

Fig. 9 Different personalized generations of the same service

from the background, while the user is using another application.

Figure 9 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 personalized according to the obtrusiveness level. On the one hand, for Bob (left branch) represented by the persona of Figure 2, the service is in the *proactive-aware* obtrusiveness level. Through all the process described in the previous steps, the service is presented by means of a *status bar notification* (see Fig. 6). An excerpt of the generated code for the *status bar notification* and the rendering of this code is shown in the left of the figure. On the other hand, for Mery (right branch), the service is in the *reactive-slightly* obtrusiveness level because she is a different type of user that prefers to go to the supermarket without a notification. For this obtrusiveness level, a *widget* is used (see Fig. 7 to see the mappings of the models) and the generated code is shown on the right of the figure. In this way, the services are generated and personalized for each persona.

2.4 Continuous evolution

Preferences of the user could change over time entailing an evolution of the type of user to another profile. So, services should be adapted according to it. This constitutes an evolution of the services in terms of obtrusiveness. Moreover, a specific preference or environmental condition within the same type of user could change implying a change in the obtrusiveness level for a specific service. Thanks to the decoupling role that the models play in the development process, this evolution is supported by the method in an easy manner. The designers can define several transitions that determine how the type of user evolves or how the obtrusiveness level for a service within a type of user evolves. A transition is composed by a *condition* and an *action*. When a condition is fulfilled, the user is evolved to another profile or the obtrusiveness level is modified by changing the attention level, the initiative level, or both, as defined by the *action*.

First a change requirement is detected. This means that the obtrusiveness level for a service is not the most adequate for a specific user. This change can be de-

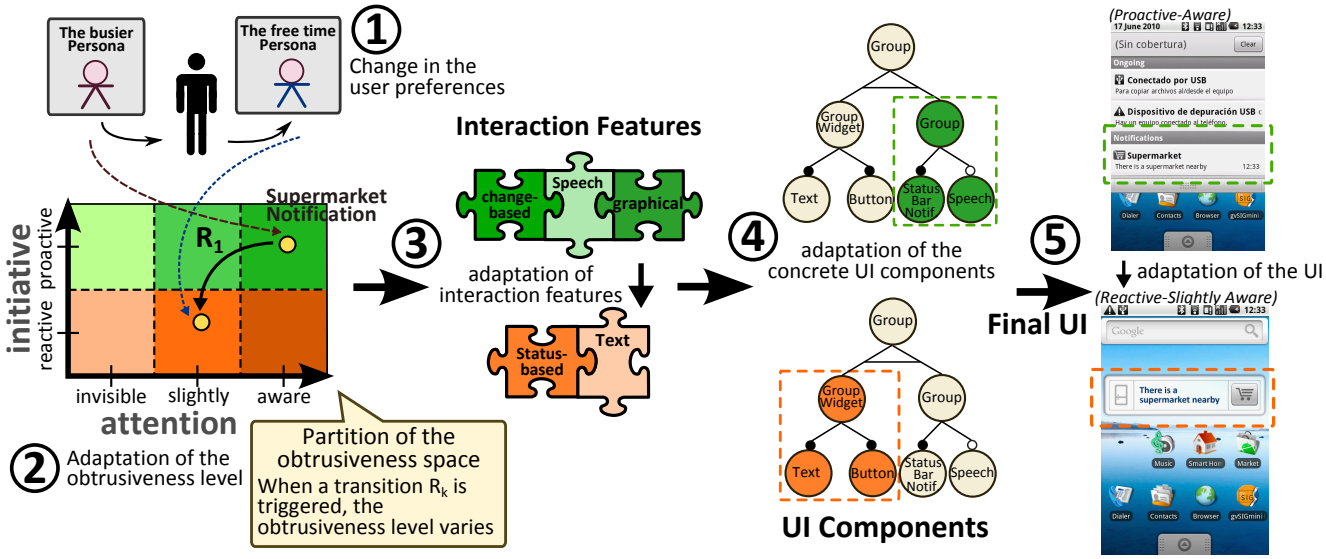

Fig. 10 Service evolution

tected through two ways. On the one hand, the user can explicitly set up his/her new preferences by choosing their new profile by means of an end-user tool similar as the developed in [39]. In this way, the system adapts services according to the new profile. On the other hand, services can be adapted in response to the fulfillment of context conditions at run-time. In order to do this, a context monitor can be used as the one we developed in the Model-based Reconfiguration Engine (MoRE) [10].

MoRE is a reconfiguration engine that provides self-configuration capabilities to a system. The way to use MoRE in this approach is similar as we defined for the adaptation of mobile business processes in terms of obtrusiveness according to the business context [20]. Using MoRE, designers can provide adaptation rules in order to indicate when the system should be reconfigured. In this way, the system can detect a context change that triggers a rule and evolves the profile of the user adapting the obtrusiveness level of services. This change in the obtrusiveness level entails the activation of another interaction features and the composition of the correspondent concrete user interface components. Thus, for example if the user has a profile in which the system informs the user *proactively* in a *slightly-noticeable* manner about the items to buy in the supermarket but the user does not reacts to it, the increasing of the items to buy (context condition) can produce an evolution of the service (due to the fulfillment of the context condition) to another obtrusiveness level that produces a notification in a more notorious manner.

We chose MoRE since it is a generic engine that can be customized by means of models. In order to use it in a particular application, designers must provide the adaptation rules and the architecture description by means of models. Thus, using MoRE, we can adapt to the change of preferences in an optimal way. To support our approach, we should use the mentioned transitions as the adaptation rules that trigger the changes in the obtrusiveness. However, the way to define these adaptation rules falls out of the scope of the present paper.

The context monitor used by MoRE can also be used to gather user information continuously in order to adapt UI obtrusiveness based on user behavior. Analyzing the user behavior (based on user's reaction), the user personality can be better understood improving the UI obtrusiveness. This can be achieved by means of adding inference rules in the context monitor, which can automatically update the obtrusiveness level according to the captured user's behavior information. Support for self-learning capability will be dealt with in further work.

Figure 10 shows an example of an evolution of the user to another profile due to a context change. This evolution implies the adaptation of the supermarket notification service according to the new profile. For this particular example, the service was in a *proactive-aware* space due to the preferences of the user described in the busier persona model (see Figure 2). For this region in the obtrusiveness space an explicit notification was used demanding a high attention from the user. With the change in the preferences or needs of the user (detected from a context monitor or set up by the user manually) the user is defined by the free time persona and the service is adapted to another obtrusiveness level (steps 1 and 2). For this region in the obtrusiveness space, subtle interaction was preferred, activating the features

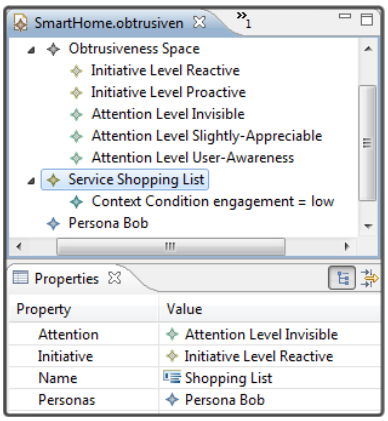

Fig. 11 Model edition support

status-bar and *text* (step 3) and producing a different personalized UI (step 4 and 5).

3 Tool support

In order to support the proposed method and allow the code generation, tool support is provided. This tool allows designers to specify (1) the obtrusiveness level for each task in the system for each persona and (2) the interaction requirements following the feature-based approach proposed in this work.

Eclipse tools are defined by combining a set of plug-ins with different functionalities. We have developed some plug-ins to support the modeling of the personalized services, 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 automatic development. The modeling community has developed several projects to support the Model Driven Engineering (MDE) paradigm under the Eclipse Modeling Project[3]. EMF permits the definition of custom modeling languages and the automatic generation of editors to support the model creation. Figure 11 shows the editor that support the definition of the obtrusiveness space model using the EMF capabilities.

3.1 Graphical editors

We have implemented a graphical editor tool that is based on Eclipse. 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. The developed tool incorporates a palette of Android components that can be labeled and linked with other components. Figure 8 shows the modeling environment. The components defined are the ones detailed in the previous section.

To model interaction features we have used Moskitt Feature Modeler (MFM). MFM is a free open-source tool that is part of the Moskitt modeling suite[4]. MFM is defined as a set of plug-ins that we could incorporate to enhance our tool support with feature modeling capabilities. MFM provides features that are well suited for the use we are making of feature models. MFM is based on the generic formalization of the feature model syntax defined by Schobbens et al. [38]. 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, since we have all the possible interaction components for the different contexts and not only the set for a specific context.

4 Validation of the proposal

In order to put in practice our proposal and validate it, we have defined a scenario within the Smart Home environment that illustrates how interaction can be personalized and adapted to provide an adequate obtrusiveness level in an ubiquitous computing environment. The adaptation takes into account the user needs and their preferences such as the message urgency, and the context conditions that affect the user such as the user location. All these factors have an effect in the obtrusiveness level to be provided. Then, we present the evaluation of the user acceptance for the interaction personalization developed in the scenario following our approach.

4.1 Smart Home case study

We applied our approach to a case study of a Smart Home environment based on the scenario of service adaptation we developed in [10]. 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.

The case study described two similar scenarios. In each scenario, services of a Smart Home were personalized according to a persona (unique for each scenario). In particular, both scenarios described a normal day for

[3] http://www.eclipse.org/modeling/

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

	Bob	**Mery**
1. Shopping list	(reactive, invisible)	(reactive, aware)
2. Meeting/lesson notification	(proactive, aware)	(proactive, aware)
3. Video recorder	(reactive, invisible)	(reactive, aware)
4. Water plants reminder	(proactive, slightly)	(proactive, slightly)
5. Supermarket notification	(proactive, aware)	(reactive, slightly)
6. Items suggestion	(proactive, slightly)	(proactive, aware)
7. Video recorder reminder	(proactive, slightly)	-
7. Water plants reminder	-	(proactive, aware)
8. Clean pool	(proactive, invisible)	-
8. Video recorder	-	(proactive, invisible)

Table 2 Obtrusiveness level of services for each persona in the case study

a particular persona (Bob and Mery) and the way interaction mechanisms of different home services changed depending on their needs, but the services presented in both scenarios were the same. In this way, users could evaluate the personalization.

Regarding the profile of Bob, the scenario presented was the following: 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 ran 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 a 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. During the meeting, the smart home reminded Bob about watering the plants. Because of watering the plants was not urgent for him, the notification appeared in a subtle manner suggesting him if he wanted that the system water the plants. When he was going back to home, he was nearby of a supermarket and the mobile notified him about it in order to optimize his time, 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. When he arrived at home, he put the mobile to charge. While it was charging, pool was cleaned automatically.

Regarding the profile of Mery, the scenario adjusted at her needs was: Mery lives in a smart home with garden. She is a housewife and everyday she gets up at 7:30 a.m. and drinks milk for breakfast. One day during the breakfast, Mery ran out of milk and she added this item to the shopping list. Then she was watching the TV and the system reminded her that she had a painting lesson and she had to leave the house. She activated video recorder. During the lesson, the smart home reminded her about watering the plants. Because of she was engaged in other activities more important, the notification appeared in a subtle manner. Before she was going back to home, she went to the supermarket. When she was there, the system reminded her the items to buy and showed her the floor map of the supermarket. When she was buying, the mobile suggested her again to water the plants, but this time in a more explicit manner (because plants was important for her and she was not engaged in an important activity). When she arrived at home, she put the mobile to charge. While it was charging, a TV program was recorded automatically.

Table 2 shows the obtrusiveness level of the different services in the case study for each persona depending on their needs. In these two scenarios, several services are presented at different obtrusiveness level. For example, video recorder service for Bob is presented first in a *reactive-invisible* manner because it begins to record automatically in reaction to the user leave. But then, the same service *proactively* notifies the user about to record the program in a *subtle* manner. In this way, users could evaluate the adaptation.

We developed a prototype version for the system described and conducted an experiment[5]. The experimental setup included an HTC Magic mobile device running Android Operating System. The experiment showed that by following our technique, personalized services with the properly interaction mechanisms in terms of obtrusiveness can be obtained.

4.2 Questionnaire and participants

To evaluate the user acceptance of the system and determine whether the interaction has been personalized and adapted properly, we used an adapted IBM Post-Study questionnaire [26] in conjunction with the ques-

[5] Screenshots of the developed prototype are showed in http://www.pros.upv.es/labs/projects/interactionadaptation

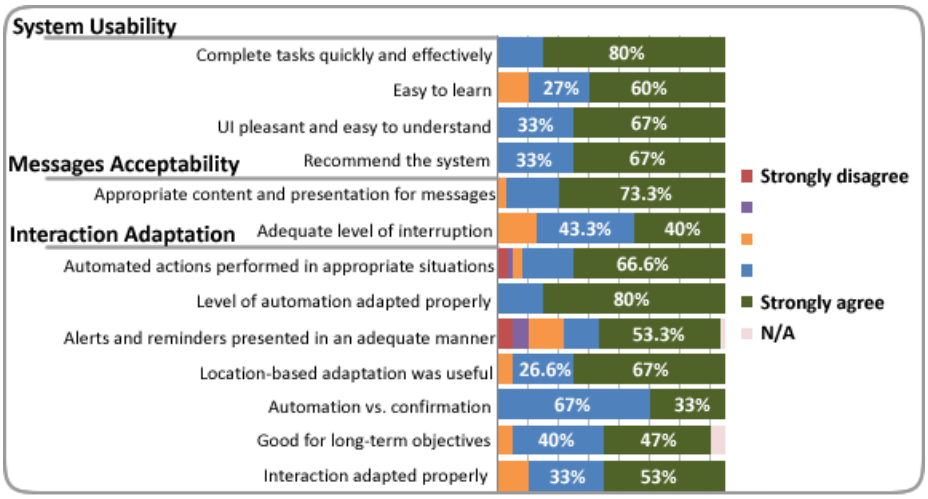

Fig. 12 Summarized results

tionnaire defined by Vastenburg et al. in [43]. On the one hand, IBM Post-Study is a questionnaire that measures 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 according to user needs*
- *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, the needs of the user and the user activity at the time of notification; and finally, the third dimension is about users' satisfaction in the interaction adaptation. We also included a NASA task load index (TLX)[6] test. This test 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] http://humansystems.arc.nasa.gov/groups/TLX/index.html

4.3 Procedure

For the evaluation of the Smart Home prototype, users adopted both Bob's and Mery's roles 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 infrastructure. An HTC Magic mobile device running Android Operating System was used to interact with the Smart Home services.

4.4 Evaluation results

Figure 12 shows a summarized table of the obtained results[7].

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 according to user needs*, 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. Although participants had to adopt the personas roles and adjust to personas needs, this is difficult when they really have another needs. In the study made by Vastenburg et al. [43], 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

[7] The complete dataset can be downloaded from http://www.pros.upv.es/labs/projects/interactionadaptation

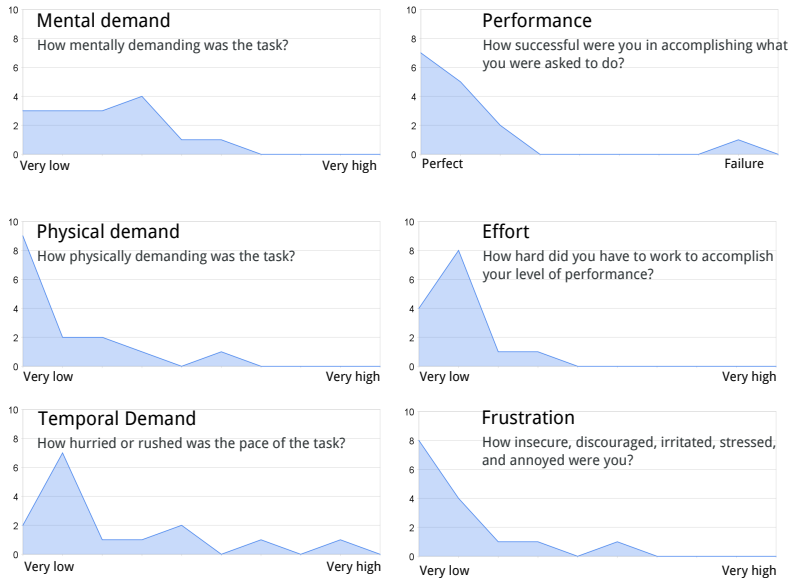

Fig. 13 Results from the Nasa TLX

of the different messages was considered appropriate 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 appropriate situations for each persona and helped 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 be able to postpone it" or "When the system clean the pool do not inform the user about that". Although the adaptation provided was considered adequate for each scenario (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 (in case of Bob's scenario), and the suddenly change of the outdoor to an indoor map of the supermarket made some users (33%) feel they were loosing control.

The initial results obtained show that by following our approach we can adjust the obtrusiveness level for the services in a detailed manner providing a good personalization. Nevertheless, additional experimentation would be required to analyse the adaptation during longer periods. Due to time constraints, we gave the users a script to follow to reproduce specific tasks and contexts of use. Using a script that was conformant to the process rules did not allow to evaluate the system in a more realistic context where services are competing with daily activities.

4.4.1 Workload

The results on workload are showed in Figure 13. 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 except 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. They show that users did not understand the question very well.

5 Related work

This work is placed in the intersection of *Context-Aware Computing* and *Considerate Computing*. In the following subsections we compare details of our work with the existing in the mentioned areas.

5.1 Context-aware computing

There are many proposals in context-aware computing that extend the models used for describing UIs in the Software Engineering in order to take into account the context of use and make UIs to be context-aware.

Calvary et al. in [9] describe a development process to create context-aware user interfaces and they give an overview of different modeling approaches to deal with user interfaces supporting multiple targets in the field of context-aware computing [34]. Our proposal introduces the notion of features to relate them in a decoupled fashion. In this way, we can describe UI adaptation over multiple platforms (1) in a declarative manner and (2) taking into account different aspects such as obtrusiveness.

Interaction concepts have been reflected in different modeling languages that are focused on the description of interaction. Van der Bergh in [5] proposes the extension of UML by means of profiles to cope with the modeling of context-aware user interfaces. Other approaches such as UIML [1], UsiXML [27] or XIML [35] define domain-specific languages that are specifically designed from the beginning to deal with the description of user interfaces in a device-independent manner. All these approaches consider pre-defined set of context factors and do not decompose them to exploit their commonalities and differences as our proposal does by means of decomposing them into features.

The modeling of interaction becomes really powerful when the descriptions can be used to guide the development of the final system in an automatic way. Despite the limitations in the automatic generation of user interfaces [32], tools such as Teresa [31] or DynaMo [12] deal with the generation of interfaces that are focused on the support of multiple platforms and contexts.

All of the mentioned approaches consider the context of use by three classes of entities: *user*, *platform* and *environment* [9] [14] [3]. In this work, we introduce the concept of obtrusiveness to consider user attention in the personalization and adaptation process. We address a different issue that is more related to human limitations of the user (e.g., attention) than technical limitations of the device (e.g., screen size).

5.2 Considerate computing

Since human attention is a scarce resource, services should be presented to users in an unobtrusive way to avoid overloading them. Attention can be viewed as a limited resource that can be modeled according to the user goals [29] [24]. The attentive user interface paradigm [44] and the considerate computing paradigm [18] aim at avoiding overwhelming the user by adapting the services based on sensed user attention. The different approaches are mainly focused on detecting or inferring attention, calculating the cost of interruption in order to predict acceptability. As early pioneers in this area, Horvitz et al. [24] demonstrated the potential use of Bayesian networks for computing the cost and value of interruptions. These approaches provide conceptual frameworks to obtain design guides, but they lack of tools for the development of this kind of user interfaces and the easy user interface definition. Conversely, our approach provides mechanisms for the design and development of user interfaces in a declarative manner in terms of obtrusiveness.

Towards creating systems that adapt their level of intrusiveness to the context of use, works focus on minimizing unnecessary interruptions for the user [36]. Hinckley and Horvitz [22] modeled interruptibility by considering the user's likelihood of response and the previous and current activity. Ho and Intille [23] compared different mental or task stages during which interruption occurs and suggested that proactive messages delivered when the user is transitioning between two activities may be received more positively. Vastenburg et al. [43] conducted a user study of acceptability of notifications to find out what factors influence the acceptability of notifications.

Given this background, our services take into account the preferences of a user in deciding the obtrusiveness level of each service. Furthermore, these initiatives are almost exclusively focused on evaluating the adequate timing for interruptions, while user interface adaptation or presentation mode has received little attention. These approaches are based on provide or not a service but they neither address the problem of adaptation of interaction nor offer tool support as we address in this work.

Adaptation to individual users and tasks is designated as *personalization* [40] [46]. Several works deal with personalizing intelligent environments [11] and adapting services to the user [7] based on the occupants presence, behavior and intentions. In these works the personalization is made at content level since user could choose what information is going to be displayed or hidden. In our approach, the personalization is made at attention level adapting the interaction mechanisms that allow users to access to service content in a different way according to the needs analyzed for each user.

6 Discussion about efficiency and usefulness

In this section we introduce a discussion of the usefulness and efficiency of the proposed method.

The usefulness of our proposal depends on the adaptation level expected (number of factors considered).

- When we handle simple applications with few adaptation factors to consider (services, users, context conditions), the definition and combination of all the models that help designers to personalize the interaction does not add too much usefulness.
- When the number of adaptation factors increase by considering many combinations of services, users and context conditions, our proposal allow to (1) have a description of the impact of the adaptation aspects and (2) reuse interaction fragments.

Android has introduced *application fragments* in Android 3.0 in order to help applications adjust their interfaces and reuse different parts of an applications user interface. This is due to the need to support more dynamic and flexible UI designs when considering different conditions such as large screens (tablets, TVs) or new interaction mechanisms. This provides a user interface composition similar as we propose in our method. The fact that a company leader in the mobile devices field opts for a fragment approximation is an indicator of the scalability and usefulness of the solution for these devices. The difference with our approach is that they are based on the technical part without dealing with adaptation according to obtrusiveness models.

Regarding the efficiency, the modeling solutions for interaction adaptation usually describe *what* information is presented to the user by means of an Abstract User Interface, 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 [9]. 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, an auditory impaired user and a noisy environment both share the auditory limitation, so the interaction with the system would be more similar in these contexts compared to the interaction offered at other user.
- *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 should 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 interaction increases with the number of context conditions considered.

In order to avoid these problems, we decompose the context conditions in their features (capabilities and limitations) represented as interaction aspects, and we use these features to describe the interaction in an abstract manner. Interaction features can be shared among context conditions to indicate their commonalities. 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 the duplication of efforts in the development are minimized since both cases are expressed as the exclusion of the auditory feature. Avoiding the duplication of efforts in the development of services we guarantee the efficiency of the proposal.

7 Conclusions

The challenge in an environment full of embedded services (where human attention is the most valuable resource) is not only to make information available to people at any time, at any place, and in any form, but to reduce information overload by making information relevant to the task-at-hand [17]. Information delivery methods should achieve the right balance between the costs of intrusive interruptions and the loss of context-sensitivity of deferred alerts [24].

This work provides an approach to define and develop personalized mobile services in terms of obtrusiveness by decoupling obtrusiveness and interaction features without duplicating efforts in the development. On the one hand, by means of personas we detect the user needs of each kind of user, define common functionalities and express them in terms of obtrusiveness. On the other hand, Feature Models allow designers to decompose interaction aspects and set the constraints for their selection. As the whole method is supported by models, feedback from users is easily mapped onto the models. Tool support has been provided by means of a MDE toolset with code generation capabilities. We developed a prototype system for a case study and conducted an experiment with end-users to evaluate it. Experimental results show that by following our technique, personalized services with the properly interaction in terms of obtrusiveness can be obtained.

Further work will be dedicated to (1) the improvement of the tool support enabling end-users to set their preferences, (2) integrate it with the Model-based Reconfiguration Engine (MoRE) [10] to achieve a dynamic reconfiguration in response to the change of user preferences and user context variations, and (3) provide

the self-learning capability of UI obtrusiveness based on user behavior to improve the adaptation.

Acknowledgements This work has been developed with the support of MICINN under the project EVERYWARE TIN2010-18011 and co-financed with ERDF, in the grants program FPU.

References

1. Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M., Shuster, J.E.: Uiml: an appliance-independent xml user interface language. In: WWW '99, pp. 1695–1708. Elsevier North-Holland, Inc. (1999)
2. Ballagas, R., Borchers, J., Rohs, M., Sheridan, J.G.: The smart phone: A ubiquitous input device. IEEE Pervasive Computing **5**(1), 70 (2006)
3. Balme, L., Demeure, A., Barralon, N., Coutaz, J., Calvary, G.: Cameleon-rt: A software architecture reference model for distributed, migratable, and plastic user interfaces. In: EUSAI, pp. 291–302 (2004)
4. Benavides, D., Cortés, R.A., Trinidad, P.: Automated reasoning on feature models. LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005 **3520**, 491–503 (2005)
5. Van den Bergh, J., Coninx, K.: Using uml 2.0 and profiles for modelling context-sensitive user interfaces. In: Proc. of the MDDAUI2005 CEUR Workshop
6. Blomquist, A., Arvola, M.: Personas in action: ethnography in an interaction design team. In: Proce. of NordiCHI '02, pp. 197–200. ACM, New York, NY, USA (2002)
7. Bright, A., Kay, J., Ler, D., Ngo, K., Niu, W., Nuguid, A.: Adaptively recommending museum tours. In: G.R. Nick Ryan Tullio Salmon Cinotti (ed.) Proc. of Workshop on Smart Environments and their Applications to Cultural Heritage, pp. 29–32. Archaeolingua (2005)
8. Brown, D.M.: Communicating Design: Developing Web Site Documentation for Design and Planning (2nd Edition). New Riders Press (2010)
9. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interacting with Computers **15**(3), 289–308 (2003)
10. Cetina, C., Giner, P., Fons, J., Pelechano, V.: Autonomic computing through reuse of variability models at runtime: The case of smart homes. Computer **42**(10), 37–43 (2009)
11. Chatfield, C., Carmichael, D., Hexel, R., Kay, J., Kummerfeld, B.: Personalisation in intelligent environments: managing the information flow. In: OZCHI '05, pp. 1–10. Computer-Human Interaction Special Interest Group of Australia (2005)
12. Clerckx, T., Winters, F., Coninx, K.: 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. ACM Press (2005)
13. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged configuration using feature models. In: Proc. of SPLC 2004
14. Duarte, C., Carriço, L.: A conceptual framework for developing adaptive multimodal applications. In: Proc. of IUI '06, pp. 132–139. ACM, New York, NY, USA (2006)
15. Evans: Domain-Driven Design: Tacking Complexity In the Heart of Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2003)
16. Favre, J.M.: 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 (2004)
17. Fischer, G.: User modeling in human-computer interaction. User Modeling and User-Adapted Interaction **11**(1-2), 65–86 (2001)
18. Gibbs, W.W.: Considerate computing. Scientific American **292**(1), 54–61 (2005)
19. Giner, P., Cetina, C., Fons, J., Pelechano, V.: Developing mobile workflow support in the internet of things. IEEE Pervasive Computing **9**(2), 18–26 (2010)
20. Giner, P., Cetina, C., Fons, J., Pelechano, V.: Implicit interaction design for pervasive workflows. Personal and Ubiquitous Computing pp. 1–10 (2011)
21. Gulliksen, J., Goransson, B., Boivie, I., Blomkvist, S., Persson, J., Cajander, A.: Key principles for user-centred systems design. Behaviour & Information Technolog **22**, 397–409 (2003)
22. Hinckley, K., Horvitz, E.: Toward more sensitive mobile phones. In: Proc. of the UIST '01, pp. 191–192. ACM, New York, NY, USA (2001)
23. Ho, J., Intille, S.S.: Using context-aware computing to reduce the perceived burden of interruptions from mobile devices. In: Proc. of CHI '05, pp. 909–918. ACM (2005)
24. Horvitz, E., Kadie, C., Paek, T., Hovel, D.: Models of attention in computing and communication: from principles to applications. Commun. ACM **46**(3), 52–59 (2003)
25. Ju, W., Leifer, L.: The design of implicit interactions: Making interactive systems less obnoxious. Design Issues **24**(3), 72–84 (2008)
26. Lewis, J.R.: Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. Int. J. Hum.-Comput. Interact. **7**(1), 57–78 (1995)
27. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V.: Usixml: A language supporting multi-path development of user interfaces. In: EHCI/DS-VIS, pp. 200–220 (2004)
28. Mao, J.Y., Vredenburg, K., Smith, P.W., Carey, T.: User-centered design methods in practice: a survey of the state of the art. In: CASCON '01, p. 12. IBM Press (2001)
29. McCrickard, D.S., Chewar, C.M.: Attuning notification design to user goals and attention costs. Commun. ACM **46**, 67–72 (2003)
30. Mori, G., Paternò;, F., Santoro, C.: Ctte: support for developing and analyzing task models for interactive system design. IEEE Trans. Softw. Eng. **28**(8), 797–813 (2002)
31. Mori, G., Paternò, F., Santoro, C.: Design and development of multidevice user interfaces through multiple logical descriptions. IEEE Trans. Softw. Eng. **30**(8), 507–520 (2004)
32. Myers, B., Hudson, S.E., Pausch, R.: Past, present, and future of user interface software tools. ACM Trans. Comput.-Hum. Interact. **7**(1), 3–28 (2000)
33. OMG: Business Process Modeling Notation (BPMN) Specification (2006). OMG Final Adopted Specification
34. Paternò, F., Santoro, C.: A unified method for designing interactive systems adaptable to mobile and stationary platforms. Interact. with Comput. **15**(3), 349–366 (2003)
35. Puerta, A., Eisenstein, J.: Ximl: a common representation for interaction data. In: Proc. of IUI '02, pp. 214–215. ACM, New York, NY, USA (2002)
36. Ramchurn, S.D., Deitch, B., Thompson, M.K., Roure, D.C.D., Jennings, N.R., Luck, M.: Minimising intrusiveness in pervasive computing environments using multi-

agent negotiation. In First International Conference on Mobile and Ubiquitous Systems pp. 364–372 (2004)
37. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley (1998)
38. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Comput. Networks **51**(2), 456–479 (2007)
39. Serral, E., Pérez, F., Valderas, P., Pelechano, V.: An end-user tool for adapting smart environment automation to user behaviour at runtime. In: Proc. of UCAmI '10 (2010)
40. Streefkerk, J.W., van Esch-Bussemakers, M.P., Neerincx, M.A.: Designing personal attentive user interfaces in the mobile public safety domain. Computers in Human Behavior **22**, 749–770 (2006)
41. Tedre, M.: What should be automated? interactions **15**(5), 47–49 (2008)
42. Unger, R., Chandler, C.: A Project Guide to UX Design: For user experience designers in the field or in the making. New Riders Publishing, Thousand Oaks, CA, USA (2009)
43. Vastenburg, M.H., Keyson, D.V., de Ridder, H.: Considerate home notification systems: a field study of acceptability of notifications in the home. Personal and Ubiquitous Computing **12**(8), 555–566 (2008)
44. Vertegaal, R.: Attentive user interfaces. Commun. ACM **46**(3), 30–33 (2003)
45. Weiser, M., Brown, J.S.: The coming age of calm technolgy pp. 75–85 (1997)
46. Weld, D.S., Anderson, C., Domingos, P., Etzioni, O., Gajos, K., Lau, T., Wolf, S.: Automatically personalizing user interfaces. In: IJCAI '03, pp. 1613–1619 (2003)