Self-adaptive unobtrusive interactions of mobile computing systems

Miriam Gil [a,*] and Vicente Pelechano [a]
[a] *Centro de Investigación en Métodos de Producción de Software,*
Universitat Politècnica de València,
Camino de Vera s/n, 46022 Valencia, Spain
E-mail: {mgil,pele}@pros.upv.es

Abstract. In Pervasive Computing environments, people are surrounded by a lot of embedded services. Since pervasive devices, such as mobile devices, have become a key part of our everyday life, they enable users to always be connected to the environment, making demands on one of the most valuable resources of users: human attention. A challenge of the mobile computing systems is regulating the request for users' attention. In other words, service interactions should behave in a considerate manner by taking into account the degree to which each service intrudes on the user's mind (i.e., the degree of obtrusiveness). The main goal of this paper is to introduce self-adaptive capabilities in mobile computing systems in order to provide non-disturbing interactions. We achieve this by means of an software infrastructure that automatically adapts the service interaction obtrusiveness according to the user's context. This infrastructure works from a set of high-level models that define the unobtrusive adaptation behavior and its implication with the interaction resources in a technology-independent way. Our infrastructure has been validated through several experiments to assess its correctness, performance, and the achieved user experience through a user study.

Keywords: Interaction adaptation, self-adaptation, pervasive computing, unobtrusiveness, mobile computing

1. Introduction

Emerging pervasive technologies have opened a new way of accessing up-to-date information (about weather forecasts, current market prices, etc.) and environmental services (a shopping service, a tourism service, etc.). This is due to the fact that nowadays the use of pervasive devices and things (e.g., smartphones, tablets, TVs, smartwatches, etc.) is widespread. Unlike desktop software (which assumes full user attention, a static view of the user environment, and stable operating conditions), mobile computing systems highlight the importance of letting the user concentrate on his/her tasks and of adapting the services proactively to changing user contexts [1].

As intelligent objects and devices are becoming part of our environment, in the near future could appear a scenario in which our fridge announces to us what recipes can be prepared with the food that is available, our TV tells us that our favorite program is beginning and asks us to record it, our mobile agenda reminds us of a meeting to attend; and all of this is happening at the same time. Clearly, living in such a pervasive environment on a daily basis may be annoying as users may frequently be inappropriately interrupted [2,3]. Conversely, if these services behave in a completely automatic or invisible manner (without requiring human input or informing the user), users can feel that their environment is out of their control, which is also undesirable [4].

With more and more digital services being added to our surroundings, they might possibly 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 [5]. Since user attention is a valuable but limited resource, pervasive services must behave in a considerate manner [6], requiring user attention only when it is actually neces-

[*]Corresponding author. E-mail: mgil@pros.upv.es.

sary. For example, a considerate library service knows that when a package of books arrives to the library, the completion of the reception task has to be done without notifying the librarian when s/he is attending customers.

In earlier work, Gil et al. [7] presented a design method to capture the degree of obtrusiveness required to interact with a service based on the user's needs. By considering the obtrusiveness aspects in a static manner, personalized interactions of services can be provided, but always with the same degree of obtrusiveness. Therefore, the user's needs and context change dynamically and the obtrusiveness aspects of service interactions should change accordingly. In this way, the obtrusiveness level of services interactions must change during runtime in accordance with changes in the user's context. For example, consider a user in an airport taking a flight and who is thinking mainly about the task of moving swiftly through the airport terminal in order not to miss the flight because s/he has arrived late. S/he probably prefers instructions that require the least attention (e.g., an arrow showing the way). By contrast, consider another user in the same airport who has sufficient time to take the flight. This second user is more relaxed and can pay much more attention (e.g., using a dialog). In principle, the mobile service interaction should be appropriately adapted to each user context automatically and seamlessly for the user at runtime.

The need to make mobile computing interactions sensitive to the user's attention is well recognized [8, 9,10]. Some studies have focused on reducing mobile interruptions by creating interruption models [11,12]. However, they have been focused more on addressing *when* to interrupt without paying much attention to the autonomic adaptation of *how* to interrupt. Some other works focused on modality preferences do not address the modality adaptation at runtime [13]. Researchers have investigated context-aware systems to adapt the phone profile [14,15], but they are limited to phone calls and are technology-dependent. Therefore, we propose to fill this gap by providing a user-centered infrastructure that is capable of adapting interaction obtrusiveness of services in a technology-independent manner. There are other approaches that rely on learning techniques to create user and interruption models [14], but they require training data and do not support cold-starts [16].

In this article, we present a **self-adaptive infrastructure** to adapt the degree of obtrusiveness required for each service interaction autonomously at runtime. Following the Autonomic Computing principles [17], our system makes decisions on its own, using high-level policies. These high-level policies are specified in design models that define how the interaction obtrusiveness varies in a technology-independent manner according to the user's context. From these models, the infrastructure determines the modifications to be performed on the interaction resources of a service in response to context changes (changes in the user's situation) in order not to disturb the user. When a context change is detected, services are retargeted to make use of the appropriate mobile interaction components in an autonomous way. Finally, we present the results of a real deployment to evaluate the correctness, performance, and the User eXperience (UX) using the system. This last evaluation reveals the relevance of considering obtrusiveness aspects in the interaction adaptation process in order to enhance the user's experience and presents some issues that need to be addressed.

The remainder of the paper is organized as follows. Section 2 presents related work. Section 3 describes the high-level models on which our system is based. Section 4 presents the adaptation process that our approach follows to self-adapt the interaction obtrusiveness of services. Then, the software infrastructure that supports the adaptation process is defined in Section 5. Section 6 describes the mechanisms that are used to adapt service interactions to the managed devices. Section 7 describes the experimentation done to evaluate the correctness and scalability of the proposed infrastructure and the achieved user experience. Finally, Section 9 concludes the paper.

2. Related work

Some studies have been conducted on context-aware mobile computing to automatically adapt the modality configuration profile of mobile devices based on context changes [18,15]. However, the focus of these studies has been on context recognition and not on the multimodal configuration as such. In this area, Korpipää et al. [19] developed a prototype tool to let users customize multimodal interactions with smart phones. However, this tool does not allow the customization of applications. Also, Barkhuss and Dey [20] conducted a study showing that users prefer autonomous behavior in spite of feeling a lack of control. Evers et al. [21] provided an extension to the MUSIC middleware [22] to define interaction practices to influence the application's behavior according to the user's perceptual focus. Their solution involve the user participation for the interaction adaptation and do not focus on the obtrusiveness aspects of the interaction.

Several architectures have also been defined in the area of context-aware mobile interaction. FAME [23] is a model-based architecture for adaptive multimodal applications. Models specify a behavioral matrix to assist in the adaptation rules development. DynaMo-AID [24] is a design process and a runtime architecture to develop context-aware user interfaces that support dynamic context changes. Aleksy et al. [25] defined an architecture that enables the dynamic loading of different elements of a mobile application, including the output components. Blumendorf et al. created MASP [26], a runtime architecture to support migration, distribution, and multimodality. Motti and Vanderdonckt in [27] reviewed research on context-aware adaptation of user interfaces and proposed a computational framework to support adaptation of the user interfaces of interactive systems. While these approaches define the adaptation space in terms of the environment and the platform, our approach defines this adaptation space for the interaction in terms of obtrusiveness. Our approach addresses a different issue that is more related to the human limitations of the user (e.g., attention) than the technical limitations of the device (e.g., screen size). Furthermore, existing solutions generally manage adaptations at the concrete user interface level, turning them into complex descriptions since they are expressed as manipulations to the user interface models. Our approach introduces the feature model to define the variability on the interaction specification. In this way, we exploit the commonalities and differences of interaction features, hiding much of the complexity in the definition of the adaptation space.

As early pioneers in the area of interruptability, Horvitz et al. [11] have gone beyond inference to highlight the crucial factors of uncertainty which should influence decisions about proactivity. Research in this area focuses on minimizing unnecessary interruptions [28] by making studies to calculate the adequate timing for them [29,30]. This research has focused primarily on determining *when* to interrupt without paying much attention on *how* to interrupt, which is what we do in this work. Their work is based on whether or not to provide a service without paying attention to the right way to provide the interaction.

In order to determine the obtrusiveness of interactions, some studies rely on automatic learning techniques. *Bayesphone* [14] is a call-handling system that learns models for attendance and interruptability and decides whether to ring the phone or transfer the call to voicemail. Rosenthal et al. [12] proposed a method to learn when to turn on and off the phone volume in order to avoid phone interruptions. These studies are limited to a single phone device. Along similar lines, Valtonen et al. [15] proposed a system to adapt the phone profile based on context changes. However, context changes are limited to time and location, and the system only works on a specific phone device. Pielot et al. [31] provided a predictor of attentiveness to mobile messages by means of machine-learning techniques. However, they only focus on instant messaging. These learning techniques also require time to adjust to new user behavior by means of training and do not support cold-starts [16]. Conversely, our infrastructure is based on high-level design models that define the obtrusiveness adaptation behavior. Therefore, interaction adaptation is performed from the first moment the system is running. In addition, since models are technology-independent, we can adapt the interaction of the various mobile devices that a user can possess, regardless of the platform.

In similar research, Apple introduced the *don't disturb* feature in iOS 6 allowing users to schedule the periods of time when the phone device is to be silenced. The fact that a leading company in the field of mobile devices has acknowledged the interruption problem and tried to make a first attempt to solve it is an indicator that more research is needed. In this direction, there are mobile apps such as Llama[1] or Locale[2] that allows managing mobile sound profiles according to the user location. However, these apps are only location-aware, not context-aware. Also they are platform-dependent. To date the research has dealt with the problem in different ways but with various limitations. There is no user-centered, technology-independent, and systematic proposal. Below, we provide details about the high-level models on which our system is based and then we present our self-adaptive software infrastructure.

3. The unobtrusive adaptation space

A key challenge to providing unobtrusive service interactions is to manage the user's attention in the adaptation process. By "unobtrusive" we mean interactions that are welcomed by the user that do not disturb

[1]https://play.google.com/store/apps/details?id=com.kebab.Llama&hl=en
[2]https://play.google.com/store/apps/details?id=com.twofortyfouram.locale&hl=en

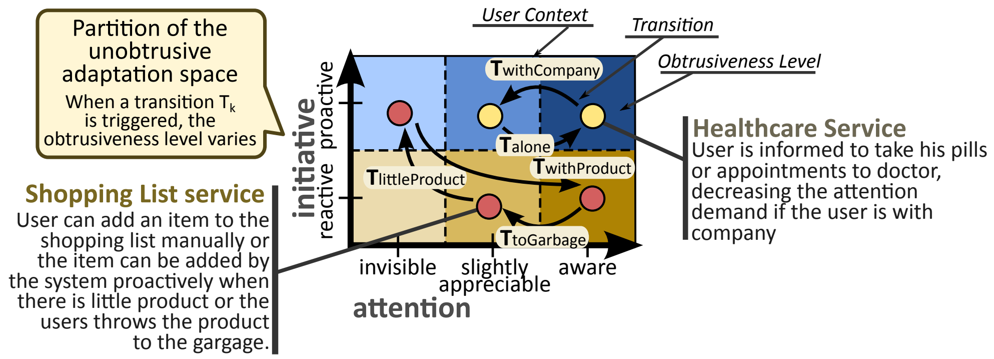

Partition of the unobtrusive adaptation space
When a transition T_k is triggered, the obtrusiveness level varies
User Context
Transition
Obtrusiveness Level
TwithCompany
Talone
TwithProduct
TlittleProduct
TtoGarbage
initiative
proactive
reactive
attention
invisible
slightly appreciable
aware
Healthcare Service
User is informed to take his pills or appointments to doctor, decreasing the attention demand if the user is with company
Shopping List service
User can add an item to the shopping list manually or the item can be added by the system proactively when there is little product or the users throws the product to the gargage.

Fig. 1. Example of the unobtrusive adaptation space for two services.

him/her. In order to address this issue, we represent the adaptation space in terms of human limitations. Specifically, we use the conceptual framework for designing unobtrusive interactions to define our *unobtrusive adaptation space* [7].

As Fig. 1 illustrates, an *unobtrusive adaptation space* is composed by *obtrusiveness levels* that describe the different possibilities in which interaction with a service can be offered. Each obtrusiveness level is formed by an *initiative level* to indicate who initiates the interaction (e.g., the user or the system) and an *attention level* to indicate the degree to which the interaction intrudes on the user's mind. Each *service* in the system is located in any of the possible obtrusiveness levels in which the service can be performed. Services in the obtrusiveness levels have *transitions* that specify how a specific service should move between obtrusiveness levels in order to adapt its interaction based on *user's context* (which is inferred from context changes expressed as rules). We consider a service as a mechanism that provides a coherent set of functionality, which is described in terms of atomic operations (or methods).

Taking the example of the services of Fig. 1, we have defined an *unobtrusive adaptation space* as described in the following (see the partition in Fig. 1). According to the *initiative level*, interaction can be *reactive* (the user initiates the interaction) or *proactive* (the system takes the initiative). According to the *attention level*, an interaction can be *invisible* (the user does not perceive the interaction), *slightly appreciable* (the user would not perceive it unless s/he makes some effort), and *aware* (the user becomes aware of the interaction even if s/he is performing other tasks). Designers can divide each axis in as many parts as they require for describing the obtrusiveness level of the services. 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. In this unobtrusive adaptation space, the healthcare service can inform the user about taking the pills proactively across different attention levels according to the user's context. Specifically, if the user is with company, interaction is provided at the *slightly appreciable* level of attention as a hint in order that only the user is aware of the alert. However, if the user is alone, interaction is provided in a more notorious manner (the *aware* level of attention). Regarding the shopping list service (to keep track of the products users want to buy), an item can be added to the shopping list *reactively* either by the user manually when s/he remembers an item to buy (*completely aware* level of attention), or in a more subtle way when the user just drops the item to the garbage (*slightly appreciable* level of attention). Also, an item can be added *proactively* by the system in an *invisible* manner for the user when the system predicts that an item is going to run out (e.g., when the fridge detects small amount of a product). It is worth noting that several services and their behaviors can be defined in the same unobtrusive adaptation space and various unobtrusive adaptation spaces can be defined with different obtrusiveness levels.

The unobtrusive adaptation space is created following a user-centered design (UCD) methodology where users take center-stage in the design [32]. In order to provide user-centered services adaptive to user's attentional resources, user preferences and needs have to be detected and analyzed. In the pre-design phase, the design team employs ethnographic field study techniques (user interviews and observations) to gather qualitative data about the potential users. The main outcome of these studies is a set of behavior patterns of users that are going to drive the definition of *Personas* [33] (user profiles). Specifically, *Personas* are powerful instruments for creating descriptive models of system-to-be users based on behavioral data that is identified during the interviews and observations. Then, from the definition of Personas, designers 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 needed and their level of obtrusiveness according to each type of user. For example, the shopping list service could be relevant for a housewife or a busy person that never remembers the products the s/he needs and a healthcare service could be needed for an elderly person who has to take pills regularly. Even though two users can need the same service, they can have different preferences about it. For example, a elderly person and a student could prefer the healthcare service in different obtrusiveness levels due to their frequency to take pills. Thus, the design of the services in the unobtrusive adaptation space would vary from user to user depending on the services needed and their preferences and priorities for each service. By establishing the degree of user attention that a service needs, we avoid developing overwhelming services. The user-centered design process is further detailed in the work of Gil et al. [7] and falls out of the scope of the present work.

3.1. Formalizing the Transitions in a Context Model

At runtime, services are performed in one of the obtrusiveness levels at a time. Thus, the level chosen at each moment depends on the current user's context. In order to offer the service interaction with the appropriate obtrusiveness level, designers must define *transitions* (described as T_k in Fig. 1) that link user's contexts with changes in the obtrusiveness level. Each transition is composed of a *user's context* and an *action*. When the user is in the specified context, the obtrusiveness level is modified by changing the attention level, the initiative, or both, as defined by the *action*. For example, the transition $T_{withCompany}$ of Fig. 1 is defined as follows (*withCompany* is the name of the user's context associated to the transition):

$$T_{withCompany} = \{(\text{Attention}, \textit{slightly-appreciable}), (\text{Initiative}, \textit{proactive})\}$$

Thus, when the user is in a context with company, the service interaction is adapted to an obtrusiveness level that demands less attention. To define and infer the different user's contexts, we use an ontology-based context model [34] that is based on the Web Ontology language[3] (OWL). OWL is a W3C standard that provides a markup language that greatly facilitates knowledge automated reasoning. The ontology-based context model used in this work is described as a collection of triples in the form of (*subject, predicate, object*). For example, (*Bob, personLocatedIn, Lab102*) means that Bob is located in the laboratory 102. This context model keeps the updated context information at runtime to reason about when to trigger an adaptation. Specifically, we took the ontology provided by Serral et al. [35] and extended it by adding the *userContext* class to properly describe the current context of the user and the relationship *currentContext* that links a person with a context. The information of this class is inferred by means of rules over the context information using the Jena Framework[4]. For example, the logic rule to infer that the user is with company is defined as follows:

```
[withCompany: (?user rdf:type pros:Person)
              (?user pros:personLocatedIn ?location)
              (?person1 rdf:type pros:Person)
              (?user pros:knows ?person1)
              (?person1 pros:personLocatedIn ?location)
              (?user pros:socialRelationships ?person1)
              →
              (?user pros:currentContext pros:withCompany)]
```

presented This rule detects that the user is in the same location of another person and they have a social relationship. Thus, when the service is in the *(aware, proactive)* level and this user context is fulfilled, the service has to be adapted to the *(slightly appreciable, proactive)* obtrusiveness level. We have a rule repository that contains a set of logic rules. Rules are manually added in the rule repository by designers, but they can also be re-defined by users by means of a *Context Specification Interface* such as the one presented by Woensel et al. [36]. This interface captures automatically the current context of the user letting the user to link the current context with a transition. In this way, new situations and contexts can be discovered and existing transitions can be refined.

3.2. Linking to Interaction Resources

Depending on the degree of obtrusiveness in which a service is performed, interaction will be offered in

[3]http://www.w3.org/standards/techs/owl (last accessed March 6, 2016)
[4]http://jena.apache.org (last accessed March 6, 2016)

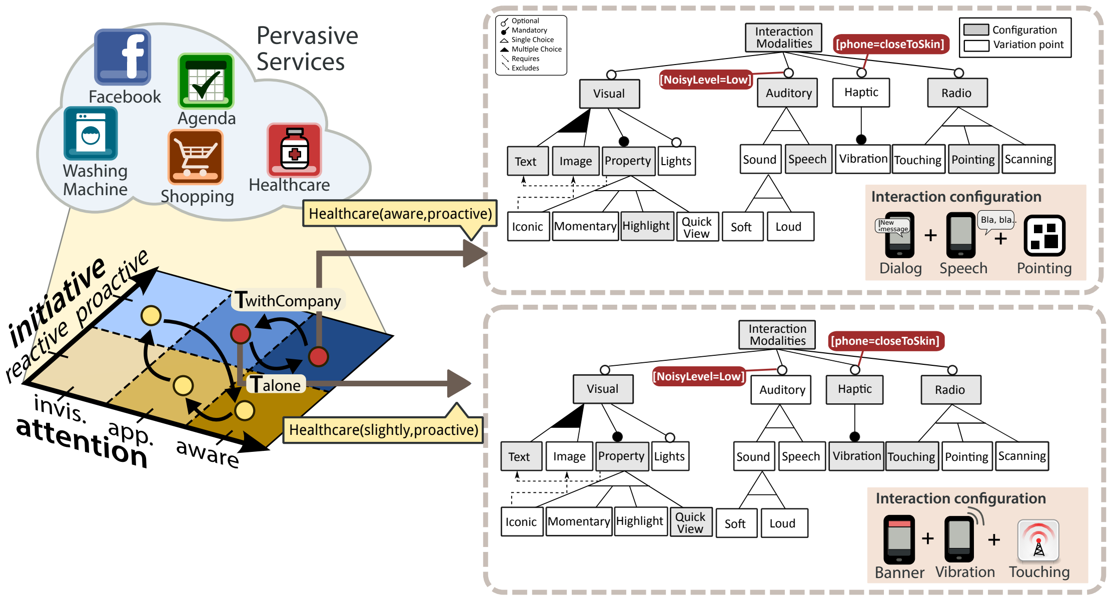


Fig. 2. Linking between obtrusiveness levels and interaction resources.

a different manner. The obtrusiveness level determines the type of interaction that has to be offered to the user. Thus, the appropriate use of a modality or modality combinations can play a crucial role, attenuating the required attention [37]. To define the interaction variability of the mobile device according to the obtrusiveness levels, we use a *feature model* (FM) [38]. We chose *feature modeling* because (1) it offers coarse-grained variability management, (2) it facilitates the representation of interaction modalities in a taxonomic way, (3) it allows us to introduce variability in the interaction specification, and (4) it has good tool support for variability reasoning [39]. Specifically, a FM is used in this work to represent the interaction modalities and modality combination as well as describe its variability. Features are hierarchically linked in a tree-like structure through variability relationships such as *optional*, *mandatory*, *single-choice*, and *multiple-choice*, and are optionally connected by cross-tree constraints such as *requires* or *excludes* (see Fig. 2).

Feature models provide an intensional description of the interaction possibilities (as opposed to an extensional description of all the possibilities) without designers having to define the interaction requirements for each user situation. In this way, we obtain common interaction aspects between user contexts. For example, the interaction provided for a user in a noisy environment shares several interaction features with the interaction provided for a user with an auditory impairment (e.g., visual modalities). We have defined our FM of interaction modalities based on existing modality taxonomies in the literature [40]. Figure 2 shows the FM for representing visual, auditory, haptic, and radio modalities and the constraints for their selection. It is worth noticing that these interaction modalities would be adapted to the requirements of each particular domain and available interaction mechanisms.

In order to specify which interaction resources support a certain service for a given obtrusiveness level, different interaction features will be enabled. We refer to the set of all activated features in a feature model as an interaction configuration (IC). For example, the grey boxes in the FM of Fig. 2 represent the interaction configuration (IC) defined to provide the interaction of the *(aware, proactive)* obtrusiveness level. The white boxes represent interaction variants that may be activated in the future to adapt the interaction to other obtrusiveness levels. For example the IC of the FM of Fig. 2 defined for the *(aware, proactive)* obtrusiveness level is assigned as follows:

$Obtrusiveness_{(aware,proactive)} = \mathrm{IC}_{Figure\ 2} =$
{Interaction Modalities, Visual, Text, Image, Property, Highlight, Auditory, Speech, Radio, Pointing }

Each configuration of the FM is defined by the set of feature states of a FM. The feasible feature states are: *active* and *inactive*. It is the task of designers to define the possible *interaction configurations* in which a FM can evolve and assign them to the appropriate obtrusiveness levels. These configurations

Table 1
Summarized information of output modalities [41,40,42,43].

Output Modality	**Visual**	**Auditory**	**Haptic**
Sensory channel	Visual	Auditory	Touch
Pros	High-specificity, supports privacy	Usable when user's focus not on screen, obtrusive/draws attention	Discreet, usable when user's focus not on the screen, reduce interruptions
Cons	User's focus needs to be on a task and screen, not usable under glaring sun	Not usable in noisy environment when privacy needed, certain social situations, obtrusive	Limited amount of information (understandability), interference, perceivability (body contact needed)
Properties	Highlighted, size, spatial relations, temporal relations, iconic representation	Volume, frequency, timbre, rhythm	Frequency, amplitude, rhythm, vibration pattern
Suitability for context	In home, office, public places, deafness	Driving, sporting, certain outdoor situation (bright sunshine), blindness	In meeting, public (noisy) places
Manifestations	Graphical icon, text, image, graph, map, lights	Beep, synthetic speech, music, acoustic alarm	Vibration, force feedback, temperature

are validated using MOSKitt4SPL[5], an open source tool for modeling dynamic software product lines that integrates the FeAture Model Analyzer Framework[6] (FAMA). FAMA enables the automated analysis of Feature Models in order to ensure a consistent interaction adaptation. In [44], more information about feature model validation can be found. Also, optional features can be tagged with context conditions indicating that the optional feature will be active if the condition is fulfilled. In this way, an interaction modality is not only chosen due to attentional factors, it is also chosen depending on environment factors. For example, if it is loud in the environment, auditory signals can not be used; or if the user is not carrying the phone close to the skin, vibration can not be used (see the tags on *auditory* and *haptic* features of Fig. 2).

The definition of configurations for each obtrusiveness level is based on the cognitive characteristics of interaction modalities [45], i.e., how information carried by different modalities is perceived by the human perceptual-sensory system. Although there is not a universal interaction technique that is well suited for any situation, there are several studies that have evaluated the effects on the cognitive load of the different modalities in order to detect the appropriate combinations of them [46,45,47]. These studies show that the selection of an appropriate interaction modality depends on (1) the user context and (2) their effects on cognitive load. In order to make our selection, Table 1 presents summarized information of the output modalities considered in this work based on the existing multimodal design taxonomies and frameworks [41,40,42,43]. This table shows the strengths, limitations and properties of the different modalities according to the literature. The three main modality types include a set of manifestations of output modalities. For example, manifestations of the auditory output include beeps, synthetic speech, music, and acoustic alarm. Each manifestation has its own features, based on which it can be identified and selected for use. The purpose of this table is to identify modalities and modality combinations best suited for different situations and information presentation needs. This table aims to help in the modeling of the interaction variability in terms of obtrusiveness. Furthermore, for the interaction between users and the physical elements, we used the comparison provided by Rukzio et al. [48] about physical mobile interaction techniques. These techniques include *touching* an element with a mobile device (using NFC or RFID), *pointing* at it with the device (using visual markers or a laser pointer), or *scanning* the environment (using Bluetooth or GPS) as considered in our variability interaction modeling.

[5]https://tatami.dsic.upv.es/moskitt4spl/
[6]http://www.isa.us.es/fama/

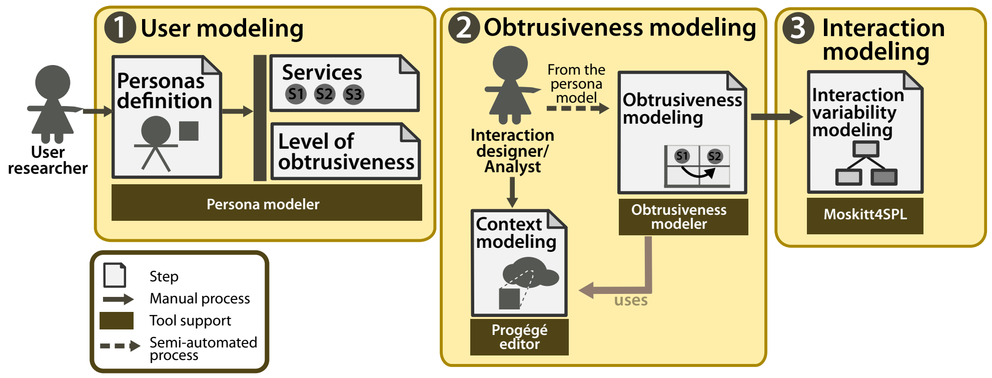


Fig. 3. Steps of the design phase.

By classifying services in this space, we adapt each service according to the user's context and attentional aspects required for each user. The main benefit of these high-level models is that they allow us to define how multi-modal interaction is adapted at runtime in a technology-independent way. It is worth noting that these models are built from user research or information-gathering methods as the ones described in [7]. Also, they are represented in XML Metadata Interchange standard (XMI)[7] and created by designers using graphical editors based on EMF[8]. An example of these models can be found on our website [49].

3.3. Unobtrusive adaptation design methodology

From a methodological perspective, designers have to perform the following steps using the provided tool support to prepare the unobtrusive services (see Fig. 3):

1. *User modeling:* Detect user needs and preferences by means of the definition of *personas* in order to determine the needed services and the obtrusiveness level required for the interaction with them. The *Persona Modeler* is used to define the personas and specify the services required and their obtrusiveness level.
2. *Obtrusiveness modeling:* Define the unobtrusive service behavior and interaction style based on user's needs and obtrusiveness to make use of the appropriate interaction resources for each context. In order to define the transitions that link user's context with changes in the obtrusiveness, the context model has to be modeled. To model the obtrusiveness, we provide an *Obtrusiveness Modeler* tool. Using this tool, the unobtrusive adaptation space is partially generated from the persona specification and manually completed using this graphical tool. The *Protege-OWL* editor is used for modeling the context in a graphical manner.
3. *Interaction modeling:* Define the interaction resources most appropriate for each service interaction in each user's context. *MOSKitt4SPL*, a free open-source tool, is used to create this model.

The provided tools help designers in the specification of the system, facilitating the adaptation description and the integration of the obtrusiveness with the service interactions. More information about the tool support, examples of models, some screenshots and a video can be found in [50],[51].

Thanks to the use of models, we can define ubiquitous services that are capable of adapting their interaction in terms of obtrusiveness by working with abstract concepts. By abstracting technical details, we can describe how the interaction varies in terms of obtrusiveness regardless of the particular technology and platform of the different ubiquitous devices. Also, by using models, we can deal in a centralized way with aspects such as obtrusiveness that are much harder to deal with in the final software system since they are spread across different parts of the code. A distinguishing aspect of our approach is the separation of concerns such as obtrusiveness and interaction. The obtrusiveness adjustment and the interaction specification are faced from a modeling perspective. By linking these aspects, when a service obtrusiveness varies, the interaction of the service is re-targeted to make use of

[7]http://www.omg.org/spec/XMI (last accessed March 6, 2016)
[8]http://www.eclipse.org/modeling/gmp/ (last accessed March 6, 2016)

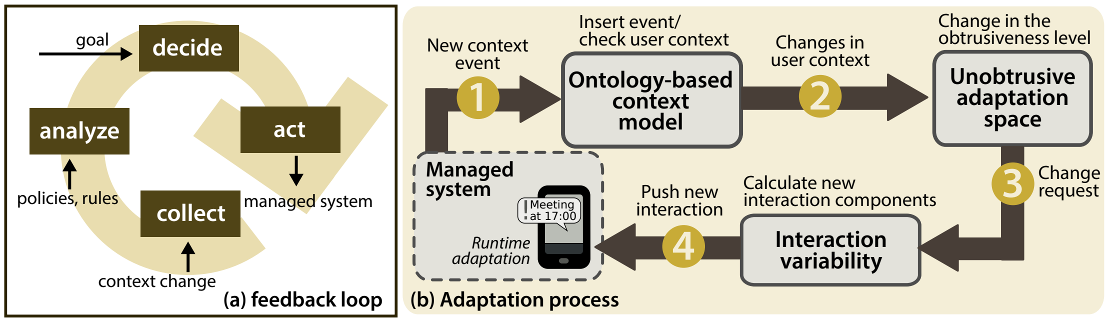


Fig. 4. (a) feedback loop and (b) adaptation process.

the new interaction components in an automated fashion. To summarize, the main benefits of our modeling method are:

- **Focus on each aspect.** Separation of concerns is promoted by our approach in order to allow designers to focus on a specific aspect at a time. Designers can define the way in which obtrusiveness is adapted according to the user's situation without thinking on the interaction mechanisms, and later, the appropriate interaction mechanisms can be chosen to cope with the obtrusiveness requirements.
- **Explore the solution space.** The used models capture not only a specific solution but also the rationale behind it. In this way, alternative solutions can be re-considered and the design knowledge can be reused for similar domains. Moreover, support for traceability between all the models allows to easily identify the interaction elements affected when the obtrusiveness degree of a service varies.

4. The self-adapting system

The high-level models previously presented capture the knowledge required to provide unobtrusive service interactions. However, in order to allow autonomous adaptations (self-adaptive), the pervasive services must themselves adapt their interactions according to the current context of the users at runtime. Autonomic Computing research [52] is the foundation to achieve application adaptation (in our case interaction adaptation). In autonomic systems, the human operator defines general policies and rules that guide the self-adaptive process, instead of controlling the systems directly. In our solution, these policies are the high-level models defined at design time.

Specifically, our approach follows a feedback loop that controls the dynamic behavior and comprises four activities: collect, analyze, decide, and act [53] (see Fig. 4(a)). The feedback cycle starts with the *collection* of relevant data about the user's context. Next, the system *analyzes* this data to check about the current state of user's context. Then, the system makes a *decision* about how to adapt the interaction of the services in order to reach the desirable state (not to disturb the user). Finally, the system *acts* by applying the interaction adaptation to the managed system (mobile device). In particular, this adaptation process follows four steps as depicted in Fig. 4(b):

1. **Collecting context changes.** The adaptation process is triggered when the system senses a relevant contextual change from the environmental and mobile sensors. The new context event is inserted in the ontology to reflect the user environment. Then, the user context is analyzed to check whether the user's current context has changed and whether an adaptation needs to be made (e.g., the user goes from being alone to being with company). This inference is based on the logic rules that use the context information that is updated in the ontology. For example, a change in the current location could mean that the user is with company.
2. **Analyzing the attentional demand.** The second step of the adaptation process is triggered when there is a change in the user context. This change can trigger an adaptation of the obtrusiveness level for a service. Thus, the unobtrusive adaptation space is checked to see if any transition depends on that new user context. For example, a new user context, the *UserWithCompany*, can trigger a change in the obtrusiveness level demanding less attention. A change request with the new obtrusiveness level is generated.

3. **Deciding the interaction mechanisms.** The third step of the adaptation process is triggered when a change request with the new obtrusiveness level is created. This change request will require new interaction mechanisms to be used by a service. Thus, the interaction mechanisms of the old and the new obtrusiveness level are compared and the necessary modifications to the interaction resources are calculated in terms of *interaction increments* (interaction mechanisms to activate) and *interaction decrements* (interaction mechanisms to deactivate) according to the configurations of the feature model. Specifically, the *InteractionIncrement* operation is made up of the interaction mechanisms in the new obtrusiveness level that are not in the current obtrusiveness level (set-theoretic difference), and the other way round for the *InteractionDecrement* operation.
   The results of the *InteractionIncrement/Decrement* operations given the adaptation of the healthcare service when the user is with company (*withCompany* context) are as follows (see configurations in Fig. 2):

   $InteractionIncrement_{withCompany} =$
   $\{$Quick View, Haptic, Vibration, Touching$\}$
   $InteractionDecrement_{withCompany} =$
   $\{$Image, Highlight, Auditory, Speech, Pointing$\}$

   These operations indicate how the services should adapt their interaction in order to move from one interaction configuration to another. For example, in the case of the adaptation of the healthcare service from the *(aware, proactive)* level to the *(slightly appreciable, proactive)* level, the *highlight*, *image*, *speech* and *pointing* resources are no longer used; instead, *quick view*, *vibration* and *touching* interaction resources are used (as calculated by the operations).
4. **Adapting the interaction components.** When the new interaction resources are calculated according to the current context, the system applies the adaptation in the pervasive device where the service is running. However, before applying the adaptation, the concrete interaction components of the target device need to be calculated. As the adaptation is performed in a technology-independent manner, we need a mechanism to translate the abstract interaction resources into the concrete interaction components of each platform. This is done by querying the *interaction providers* of each device (a component explained in the next section) to realize the mappings between the abstract resources and their related interaction components.
   For example, if our infrastructure indicates that a quick view with vibration is needed for service S, this operation replaces these components with the ones that match the specific target technology (e.g., in an Android system, the banner is translated into the status bar component).

In this section, we have illustrated how the autonomic reaction of a system can be calculated by taking the high-level models as a basis. In the next section, more detail is provided about how the required steps are supported by our infrastructure.

5. Adapting the interaction obtrusiveness at runtime

In order to carry out the adaptation process, we provide an software infrastructure (AdaptIO) that enhances services with interaction obtrusiveness adaptation capabilities.

Our infrastructure provides the following benefits:

– It is available to the pervasive and mobile services in a centralized manner.
– It senses context changes from environmental and mobile sensors.
– It is based on the high-level design models in order to adapt service interactions based on user's context in a technology-independent manner.

5.1. The self-adaptive software infrastructure

In order to make interaction obtrusiveness adaptation a reality, we define a model-based infrastructure, i.e., AdaptIO. AdaptIO is a software infrastructure to support the considerate interaction adaptation of services at runtime in the pervasive computing domain. To achieve this, it senses the context and adapts the interaction resources of each service of the managed system in terms of obtrusiveness. To enable autonomic behavior on the managed system, our infrastructure is based on the IBM reference model for self-management [54], which is called the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) loop. In our case, the knowledge is represented in the high-level models introduced in Section 3.

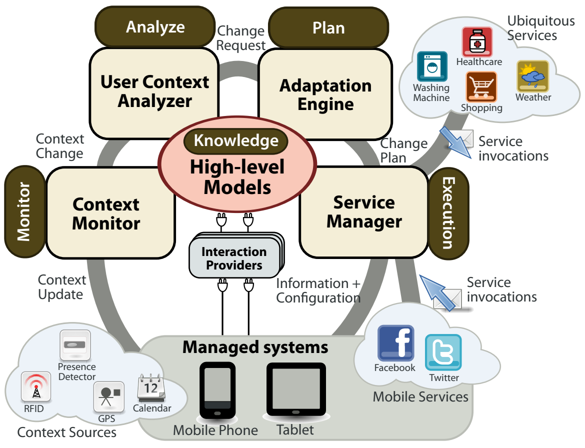


Fig. 5. Infrastructure component overview

Our design goal for AdaptIO was to support the interaction obtrusiveness adaptation in a *modular* and *extensible* manner by decoupling context processing, technology-independent interaction adaptation, and service management. We also provide a set of components in a pluggable manner to address the technological heterogeneity of pervasive and mobile devices. Figure 5 illustrates the AdaptIO infrastructure components and their connection to context sources, services, and managed devices (managed systems). For the definition of the infrastructure, we rely on the *component* concept since this is a well-understood concept that can be implemented in most of the implementation technologies available. Components are the basic pieces that make up the system.

AdaptIO is designed to provide a loosely-coupled and model-based solution. It allows us to define how the different services are integrated (by means of a *Service Manager*) and adapted (by means of an *Adaptation Engine*) to all the changes in the user context (by means of a *Context Monitor*) in terms of obtrusiveness (by means of the *User Context Analyzer*) using the design models. In the following subsections, more details are given about these components and their supported operations (with some implementation details).

5.1.1. Context Monitor

The *Context Monitor* is the component that is in charge of the monitoring process. It detects changes in the user context and translates them to context events. For example, the detection of a RFID tag with the X id means that Bob has been detected. These context events are inserted in the ontology-based context model to reflect the user environment. Note that this update must be performed at runtime. Then, the context change is passed to the *User Context Analyzer*.

Context changes are physically detected by environmental (physical sensors) and mobile sensors (virtual sensors), which are controlled by pervasive and mobile services. Therefore, in order to capture context changes, the monitor continuously monitors the execution of the services. Specifically, the pervasive services that control the physical sensors are developed using the development method presented by Serral et al. [35], which allows us to automatically generate Java/OSGi[9] pervasive services from high-level abstraction models. Using OSGi, the *Context Monitor* can listen to the changes produced in the services to detect context changes. The mobile services that control the virtual sensors are implemented as background services running on the mobile devices that monitor changes in their own sensors. For example, the user location is detected by the GPS service that controls the mobile GPS sensor and presence detector devices of the environment. Specifically, we have two *Context Event Listeners*: one in charge of listening to events from the environmental sensors and another in charge of listening to events from mobile devices.

5.1.2. User Context Analyzer

When a context event is detected, the *User Context Analyzer* analyzes the context change to infer the new user's context and to determine whether a service interaction adaptation needs to be made. For example, a change in the user's location could mean that the user is working or with company. If the user context has changed, the *User Context Analyzer* generates a change request with the new user context and passes it to the *Adaptation Engine*.

In order to accurately infer the user's context, the *User Context Analyzer* is based on logic rules. These logic rules use the context information updated in the context model (see an example of a rule in Section 3). We have a rule repository that contains a set of logic rules. Rules are manually added in the rule repository by designers in the design and runtime phase.

The main operations that support the *User Context Analyzer* are: the operation for loading rules from a folder, the operation for inferring and getting new data, and the operation for updating the user context changes in the ontology.

[9]http://www.osgi.org/ (last accessed March 6, 2016)

5.1.3. Adaptation Engine

When alerted by the *User Context Analyzer* of a change request, the *Adaptation Engine* consults the unobtrusive adaptation space to check if any transition of the current obtrusiveness depends on that new user context. If so, the engine calculates the necessary modifications to the interaction resources and generates an adaptation plan. This plan represents the set of high-level interaction changes for each service.

In order to calculate the necessary modifications to the interaction mechanisms by means of increments and decrements (explained in Section 4), the adaptation engine queries the obtrusiveness and interaction model at runtime. This is based on four main operations: (1) the *getTransitionsToCheck* operation, (2) the *checkUserContext* operation, (3) the *triggerTransition* operation, and (4) the *getDifferences* operation.

The process that the *Adaptation Engine* follows when it receives a change request with the updated user context is the following:

- For each service, the *Adaptation Engine* gets the output transitions of the current obtrusiveness level of the service. This activity is supported by the *getTransitionsToCheck* operation.
- Then, the *Adaptation Engine* checks the transitions to see if any user context has been fulfilled according to the user's current context (this means that an adaptation needs to be made). This is supported by the *checkUserContext* operation.
- For the transitions that have been fulfilled, the *Adaptation Engine* aggregates their actions to calculate the target obtrusiveness level and avoid inconsistencies when the transitions are triggered. The median is used to obtain an average movement in the unobtrusive adaptation space if more than one transition is fulfilled. We use the median instead of the mean in order to prevent extreme results from affecting the changes in the obtrusiveness level.
- Then, the *Adaptation Engine* applies the final action by obtaining the current interaction configuration and the target one and calculating the difference between both configurations. This difference is calculated by means of increments and decrements with respect to the current configuration. These procedures are supported by the *triggerTransition* and the *getDifferences* operations.

The plan of high-level interaction increments/decrements is notified to all of the installed *Interaction Providers*. Since the adaptation is performed in a technology-independent manner (high-level concepts), we need a mechanism to translate the abstract interaction resources to be used into the specific resources for each platform. To do this, we define the *Interaction Providers*, which are pluggable components in charge of converting the high-level interaction plan into a specific one with the specific interaction components of the underlying managed systems. For example, the *Quick View* is translated to a *Status Bar* on Android and a *Banner* on iOS. Finally, the plan of low-level components is passed to the *Service Manager*. The operation in charge of supporting this translation is the *doWeaving* operation.

5.1.4. Service Manager

The *Service Manager* is the component that is in charge of applying the adaptation to the underlying system. When the *Service Manager* receives a change plan that contains the interaction components that have to be adapted for a service, it stores the plan and waits until a service has to interact with the user. Typically, plenty of local and remote services or applications are running on a user's mobile device (e.g., agenda), which may interact with the user to notify him/her of important events or information. These services are registered in the *Service Manager* and invoke it when they have to interact with the user. Thus, when a service invocation is received (i.e., a notification, information that is useful for the user, etc.), the *Service Manager* retrieves the interaction components to be used from the change plan, composes the interaction, and pushes the information to the user's managed device with the appropriate interaction configuration.

The *Service Manager* can also be queried in order to retrieve the service information based on certain criteria (e.g., read last information, retrieve all notifications, retrieve deleted information, etc.). Since in a mobile environment users can change from device to device on the go, it is desirable to allow users to access their service's information from a single application point. The *Service Manager* component provides a unified view of the information for all the services in which the user is involved. This distributed schema is common in mobile approaches. Two operations are implemented:

Receive service information. To receive data from the services, a web service has been implemented. This way, services can send any kind of data to the *Service Manager* simply by sending the information via HTTP.

Execute a service adaptation. Once a service invocation is received, it is adapted to the user's current situation and sent to *Managed Systems* (e.g., mobile phones). To do this, our infrastructure makes use of push notifications (remote notifications). The information processed is pushed to the *Managed Systems* (via the Push Notification services) when the service has to interact with the user. The service information and the interaction configuration to be used are specified in the payload. This payload is sent to all of the registered devices.

6. Implementation and deployment of the infrastructure

This section provides details about the architecture used to deploy our infrastructure, describing the key design choices and solutions, and giving implementation details about it. AdaptIO follows a client/server architecture. The context processing, current user context inferring, adaptations and service information are managed at the server. *Managed systems* are the clients that rely on the server components for receiving the service information with the appropriate interaction components to be used. Therefore, the logic remains on the server side so that mobile clients do not have to deal with complex processes that may have an impact on performance.

6.1. Server-side subsystem

The components of the server side (*Context Monitor*, *User Context Analyzer*, and *Adaptation Engine*) are implemented using Java/OSGi technology. With this technology, the software infrastructure becomes an independent operative system that can be dynamically constructed from reusable and collaborative components, which are known in the OSGi terminology as *bundles*. Therefore, the infrastructure is developed to be run on an OSGi service platform. An OSGi service platform is an instantiation of a Java virtual machine, an OSGi framework, and a set of bundles.

The OSGi framework runs on top of a Java virtual machine and provides a shared execution environment to install, update, run, stop and uninstall bundles without needing to restart the entire system. To minimize the coupling among bundles, the OSGi framework provides a service-oriented architecture that enables bundles to dynamically discover each other for collaboration. An installed bundle can register services by publishing their interfaces using the framework's service registry. Thus, when a bundle queries the registry, it obtains references to actual service objects that are registered under the desired service interface. For example, each *Interaction Provider* is encapsulated in an OSGi bundle. In this way, we can add as many *Interaction Providers* as we need without having to stop the system. This provides flexibility for adding new devices (*Managed Systems*).

The framework also manages dependencies among services to facilitate coordination among them by using *Wire* objects. A Wire object acts like a communication channel between a Producer service and a Consumer service. Therefore, we use the *Wire Admin Service* in OSGi to carry out the inter-component eventing of the components of our system: the *Context Monitor* and the *User Context Analyzer*; the *User Context Analyzer* and the *Adaptation Engine*; and the *Adaptation Engine* and the *Interaction Providers*. When a component has an event to deliver, the component source calls all of the event listeners in the service registry.

To enable this communication, the Producer service must implement the OSGi Producer interface, while the Consumer service must implement the OSGi Consumer interface. There are two ways to establish communication using a wire: 1) the Producer service can send information to the Consumer service or 2) the Consumer service can request information from the Producer service. In our approach, the communication between services using a wire is always produced from the producer to the consumer.

Furthermore, OSGi enables the integration of heterogeneous devices and sensors in pervasive environments by means of the service discovery. Services exhibit external devices such as UPnP devices or KNX-EIB devices, which are the ones that we use to gather context information besides mobile sensors.

The *Adaptation Engine* uses Eclipse Model Query[10] (EMFMQ) to query the models at runtime and the EMF Compare plugin[11] to calculate the differences between the interaction configurations. EMFMQ facilitates the process of search and retrieval of model elements in a flexible, controlled, and structured manner. To achieve this, the plugin allows the construction and execution of queries in a SQL-fashion. We

[10]http://www.eclipse.org/modeling/emf/?project=query (last accessed March 6, 2016)

[11]http://www.eclipse.org/emf/compare/ (last accessed March 6, 2016)

use these queries to search for and get the instances of the model that need to be accessed or modified. EMF Compare provides a generic comparison engine that allows model comparison for any kind of EMF model. Specifically, we use the *diff* class to calculate the difference between the two interaction configurations.

As context is represented by means of OWL ontologies, to implement the operation to insert context events in the ontology (*InsertContextEvent* operation), the *Context Monitor* uses the INSERT form of SPARQL[12]. SPARQL is the W3C recommendation query language for RDF. This query language is based on graph-matching techniques. Given a data source, a query consists of a pattern that is matched against the data source, and the values obtained from this matching are processed to give the answer. The data source to be queried can be an OWL model like the one we use for the context model. To implement the rules of the *User Context Analyzer*, we have used the Jena Framework[13], which is a Java framework for building Semantic Web applications. It provides a programmatic environment for OWL and SPARQL and includes a rule-based inference engine (see an example of a rule in Section 3.1).

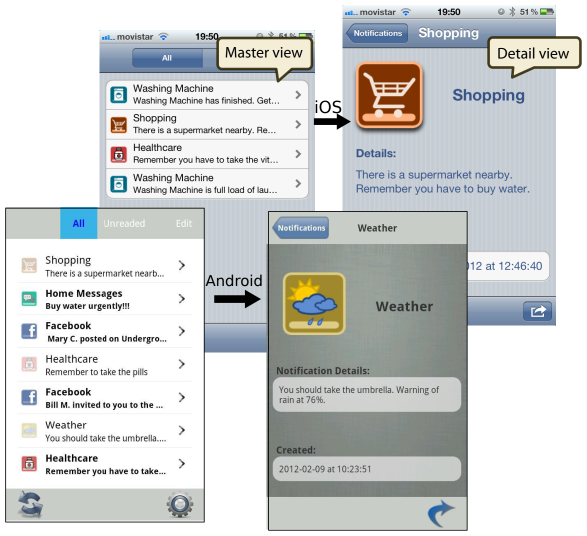
Finally, the *Service Manager* is implemented in PHP, which is a server-side scripting language that is interpreted by an Apache web server and connected to a MySQL database that stores the received interaction configurations. This way, we can keep track of the adaptation traces to be able to analyze them later and check the proper functioning of the software infrastructure. Also, these traces are shown in a web page to assist engineers in the task of checking the correct adaptations of the services (see Section 7). To implement the web service of the *Service Manager*, the Restlet Framework[14] has been used. The information to transfer has been formatted in JSON, which is a lightweight data-interchange format. Therefore, all the components understand the format of the information. Also, in order to implement the push notifications we have used Android Cloud to Device Messaging[15] (C2DM) for the Android platform and Apple Push Notification service (APNs) for iOS. Further information about the implementation is available in [55].

[12]http://www.w3.org/TR/rdf-sparql-query/ (last accessed March 6, 2016)

[13]http://jena.apache.org (last accessed March 6, 2016)

[14]http://www.restlet.org/ (last accessed March 6, 2016)

[15]https://developers.google.com/android/c2dm/ (last accessed March 6, 2016)



Washing Machine
Washing Machine has finished. Get…

Shopping
There is a supermarket nearby. Re…

Healthcare
Remember you have to take the vit…

Washing Machine
Washing Machine is full load of lau…

Shopping

Details:
There is a supermarket nearby.
Remember you have to buy water.

12 at 12:46:40

Shopping
There is a supermarket nearb...

Home Messages
Buy water urgently!!!

Facebook
Mary C. posted on Undergro...

Healthcare
Remember to take the pills

Facebook
Bill M. invited to you to the ...

Weather
You should take the umbrella....

Healthcare
Remember you have to take...

Weather

Weather

Notification Details:
You should take the umbrella. Warning of rain at 76%.

Created:
2012-02-09 at 10:23:51

Fig. 6. Implementation of the managed system on iOS and Android.

6.2. Client-side subsystem: managed systems

The client-side subsystem is formed by the *managed systems* deployed in the *managed devices* to adapt their interaction obtrusiveness. In order to validate our approach, we have implemented a *managed system* on both Android and iOS. We chose those platforms because they allow interaction components to be easily managed. On the one hand, Android provides an open application framework that supports advanced interaction techniques such as text-to-speech synthesis and easy communication mechanisms to integrate the functionality of applications. On the other hand, iOS is a closed platform where mechanisms for component interoperability and background processing are very limited. However, the APNs alleviate these problems by providing a robust and efficient service for propagating information to devices using a given configuration.

Specifically, we have implemented a mobile application on each platform following a *master-detail* interface that displays a master list of all the service information received and the details of the current selected information. When some data of a service is received at the *managed system*, the *managed system* notifies the user through the appropriate interaction components according to the current user situation. The notification is also shown in the master list. When the user selects a notification from the list, the details of the notification are shown. Figure 6 shows the views for an implementation on iOS and Android[16]. When the application is installed in the device, it registers the device to the infrastructure.

6.2.1. The managed system on Android

One of the managed systems was implemented on the Android platform. Although our approach is not platform-dependent, we took advantage of this platform to implement the adaptation actions. The Android platform provides the *Intent* messaging facility for late runtime binding between components. Moreover, it provides loosely-coupled components[17] such as *Service*, which provides functionality that is executed in the background, and *Activity*, which provides the user interface. Also, a more sophisticated user interface can be described in Android by merging different fragments defined in an XML layout file that separates the presentation from the behavior.

In order to integrate our infrastructure with the managed system on Android, a *controller* was implemented as an Android Service running in the background. This component was developed to act as a local *broker*. It supports the following operations:

Activation and deactivation of components. Android allows the dynamic activation and deactivation of components (services and activities). The different components that are executed in the mobile client are activated and deactivated by the controller as our infrastructure requests them. The *startService/stopService* and *startActivity/stopActivity* methods defined by Android are executed by the controller to activate and deactivate interaction components (implemented as services or activities). To locate a component in Android, an *Intent* is launched that describes the components to be started. *Intents* are abstract descriptions of the components to be activated (e.g., vibration feedback) regardless of the specific components that provide the functionality (e.g., vibrator resource). An *Intent* can also carry small amounts of data to be used by the component that is started. For example, the pattern to be followed or the length of time to vibrate can be included in the vibration intent.

[16]The UIs have been designed according to the design guidelines of both platforms

[17]http://developer.android.com/guide/components/index.html: Android components (last accessed March 6, 2016)

Dependency changes. Android provides loosely-coupled communication mechanisms among components. To request an external functionality, a component launches an *Intent*. In this case, the *controller* intercepts the signals from our infrastructure (abstract Intents) and translates them to more specific ones that match the specific underlying technology. For instance, if the infrastructure indicates that a *quick view* with *vibration* is needed for service S, the controller replaces the general *Intent* with one that contains the same data but with the specific data of the component description. The latter would be the component corresponding to the Android notification service that is in charge of showing status bar notifications (e.g., "*es.upv.pros.Notification.StatusBar*").

7. Evaluation

We present the evaluation of our infrastructure by means of three experiments to measure its correctness and performance and to assess the User eXperience. We first present the correctness and performance evaluation by means of objective measures and then the user experience evaluation by means of subjective measures that were collected with a questionnaire from users.

7.1. Correctness of the software infrastructure

This evaluation was based on assessing to what degree the software infrastructure (AdaptIO) performs its tasks as defined in the specification. In order to evaluate this criterion, we need to calculate if it executes the specified adaptations correctly. This implies checking whether the service interaction is adapted to the correct obtrusiveness level and determining if the new interaction mechanisms to be used are calculated correctly.

To perform this, we analyze *system execution traces* [56]. Execution traces produced by software systems during their operations capture important runtime information, and thus are valuable sources for validating software functional properties such as correctness. Data gathered from a running program has great potential for self-adaptive systems because it relies on direct monitoring mechanisms that describe the system's actual behavior [57]. Traditionally, software engineers have used code-level tracing to capture a running system's behavior. An alternative is to generate and analyze model-based traces, which contain rich seman-

Table 2
Example of a trace entry: In a Meeting

Title:	In a Meeting
Context event:	(Bob, PersonLocatedIn, Laboratory102)
Context condition fulfilled:	@inMeeting
New obtrusiveness level:	Agenda (Invisible-proactive)
Previous obtrusiveness level:	Agenda (slightly appreciable-proactive)
Reconfiguration actions:	Activations: {Iconic}, Deactivations: {QuickView, Haptic, Vibration}

tic information about the system's runs at the abstraction level that its design models define [58]. Since we use high-level models to define the behavior of adaptations, it is suitable to use model-based tracing to check the correctness of our infrastructure. Given the semantics of a trace, an engineer can check if a syntactically correct trace is consistent with regard to a specific run (is the representation correct?), and, more generally, if it is realizable. That is, if a system (and a run) exists from which it could have been generated.

To support model-based tracing, AdaptIO stores trace entries each time that an adaptation is performed. Trace entries range from the context events and user contexts that trigger the adaptations to the calculated adaptation plan when an adaptation is triggered. Since the adaptation is driven by models at runtime, our infrastructure is able to keep the trace entries at the same abstraction level as the runtime models. That is, both the runtime models and trace entries are based on concepts such as services, obtrusiveness levels, and interaction components.

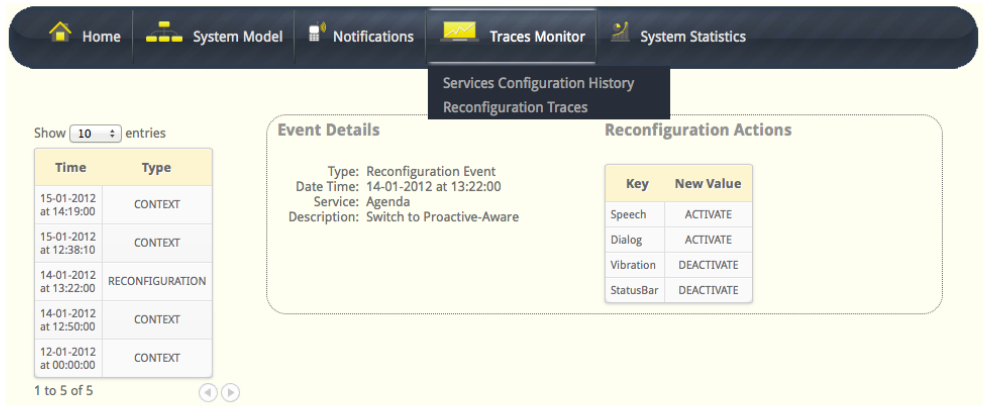
The trace entries that our infrastructure stores can belong to several entry types. We present these entry types and provide examples of their instance creation as follows (Table 2 shows an example of a trace entry for an adaptation scenario):

1. **Context event.** This entry type provides information about the context events that have been detected. Consequently, these events are the ones that can trigger an adaptation when the user context changes. This entry type also maintains the time stamp of the detected event. For example, an instance of this entry type is created when a user is detected in the Laboratory102 (*Bob, personLocatedIn, Laboratory102*).
2. **Context condition fulfilled.** This entry type provides information about a user's context that has been inferred. This fulfilled user's context triggers an adaptation. For example, an instance of this entry type is created when the *withCompany* context is fulfilled.
3. **New obtrusiveness level.** This entry type provides information about the new obtrusiveness level in which a service has to be adapted and about the new interaction configuration in terms of interaction resources. For example, an instance of this entry type is created when our infrastructure processes the user context *withCompany* and detects that an adaptation is needed.
4. **Previous obtrusiveness level.** This entry type provides information about the obtrusiveness level and the interaction configuration before the adaptation. For example, an instance of this entry type is created before the adaptation is performed to accommodate the new user context (e.g. with company).
5. **Reconfiguration actions.** This entry type provides information about the calculated adaptation actions to perform the interaction adaptation. This information is stored in terms of interaction components to activate and deactivate. For example, an instance of this entry type is created after an adaptation plan is performed and the new obtrusiveness level is reached.

In order to visualize and analyze the generated traces, we have developed a *Trace Monitor* tool. Figure 7 shows a screenshot of the main view of the Trace Monitor, displaying an adaptation trace with some of these entry types such as the new obtrusiveness level and the set of reconfiguration actions to perform the adaptation.

These trace entries provide a way to formally and quantitatively characterize and investigate the specific adaptation the trace was generated from (vertical trace) and also the overall running of the system (horizontal trace). On the one hand, vertical traces are related to a snapshot of an adaptation. They quantitatively reflect the state of an adaptation at certain time points in the execution. On the other hand, horizontal traces are related to an interval of an execution. They are evaluated over a time interval, typically a complete execution or a sequence of connected adaptations. In this work, we analyze vertical traces to check that adaptations are performed correctly.

Thus, the proposed experiment consists in analyzing that the registered traces match their specification in the models. To perform this, we have implemented a



Time	Type
15-01-2012 at 14:19:00	CONTEXT
15-01-2012 at 12:38:10	CONTEXT
14-01-2012 at 13:22:00	RECONFIGURATION
14-01-2012 at 12:50:00	CONTEXT
12-01-2012 at 00:00:00	CONTEXT

Event Details

Type: Reconfiguration Event
Date Time: 14-01-2012 at 13:22:00
Service: Agenda
Description: Switch to Proactive-Aware

Reconfiguration Actions

Key	New Value
Speech	ACTIVATE
Dialog	ACTIVATE
Vibration	DEACTIVATE
StatusBar	DEACTIVATE

Fig. 7. Screenshot of the Trace Monitor tool.

JUnit[18] test to assert that both values are equal. Specifically, the JUnit method takes a trace entry and checks the following:

1. The new obtrusiveness level calculated from a fulfilled user context is the same as the one defined in the transitions of the unobtrusive adaptation space (see Section 3.1).
2. The reconfiguration actions to be applied (list of interaction increments and decrements) are correctly calculated.

To compare the reconfiguration actions of the trace against their specification in the models, we query the interaction model and perform a set-theoretic difference between the interaction mechanisms that are associated to the new obtrusiveness level and the previous obtrusiveness level. Finally, we use the *assertEquals* method that is defined in the JUnit framework to test that the values are equal. When the arguments to assertEquals are not equal, the test fails.

We applied the JUnit test to the adaptation scenarios proposed in Section 7.3. The testing of these scenarios confirmed that the envisaged adaptation behavior was achieved; consequently, the software infrastructure interprets the models correctly. However, when analyzing the model-based traces, we discovered inconsistent behaviors in some adaptations that were initially designed because there were sink obtrusiveness levels. Therefore, traces helped us to correct the design of these adaptations. Despite this, the testing shows that our infrastructure is capable of executing the unobtrusive adaptations correctly as described in the models.

[18]http://www.junit.org (last accessed March 6, 2016

7.2. *Performance of the software infrastructure*

This experiment was based on assessing to what extent infrastructure performance could be affected by providing adaptation capabilities based on models at runtime. The use of the high-level models by the rest of the components impacts overall system performance. In particular, the incorporated latency is determined by (1) the model manipulation frameworks, and (2) the model population (including the number of services taken into account). In this experiment, we evaluated the efficiency of AdaptIO during dynamic adaptations in terms of execution time. As model manipulation frameworks, we used SPARQL to manage the context model and EMF Model Query since the unobtrusive adaptation space is represented in XMI.

When evaluating the proposed scenario, processing our designed models did not have a significant impact on performance. However, to validate whether our infrastructure scales to large systems with more services [59], we quantified this overhead for large models that were randomly generated with five hundred new elements each iteration. After the model population, the following operations were performed: *InsertContextEvent*, *CheckUserContext*, *triggerTransition*, *getDifferences*, and *doWeaving*. AdaptIO requires these operations to be efficient enough to gather the necessary knowledge without drastically affecting the system response.

Figure 8 shows the milliseconds that the infrastructure took to perform each one of the model operations. The operations with the highest temporal cost were the operations that were related to triggering transitions (*triggerTransition* and *getDifferences* operations) because they navigate through the models to obtain the

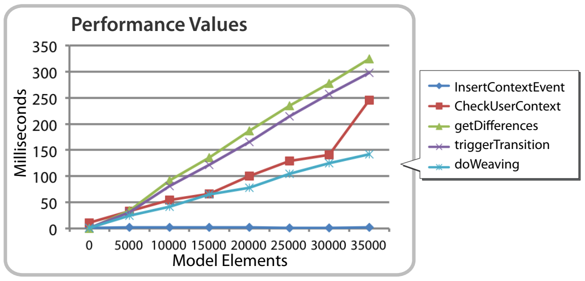


Fig. 8. Values of the infrastructure performance.

appropriate interaction resources for each context. The operation to manage context with the highest temporal cost was the *CheckUserContext* operation, which provided a reasonably fast response time (<250 ms for 35.000 elements). Although the *doWeaving* operation also operates over the models, it got less response time because it only queries the elements involved to doing the weaving (it does not navigate through the entire model). Even with a model population of 35.000 elements, all these operations provided a fast response time (<350 ms), at least for the kind of services we were addressing. Overall, the results showed that our infrastructure gathers the necessary knowledge from the models without drastically affecting the performance.

7.3. *User Experience Evaluation*

This third experiment was focused on evaluating the User eXperience (UX) when using our system (to evaluate user's feelings). User experience describes all aspects of interactions between a user and a software product, not just the ISO 9241-11 factors, effectiveness, efficiency and satisfaction. User experience introduces new concepts to the quality of software like fun, beauty and pleasure. This evaluation has become important in mobile contexts given the ubiquity and intelligent capabilities of systems of this kind. Thus, the goal of this experiment is to measure the user experience when using an implementation of a managed system with obtrusiveness adaptation capabilities. For the evaluation, we presented the users with a non-adaptive version of our system (usual notification system) and the adaptive one (our system with self-adaptive capabilities). In this way, the users could compare both systems and better measure their UX. Note that we took counterbalancing measures to avoid order effects (further explained in subsection 7.3.3). Afterwards, the users were handed an AttrakDiff 2 questionnaire (one of the most influential questionnaires to measure UX) to fill out to collect their attitudes towards both systems.

According to the Goal/Question/Metric template [60] the objective of the experiment was:

Analyze: our self-adaptive software infrastructure (adaptive system).
For the purpose of: evaluating its user experience.
With respect to: a traditional system without adaptive capabilities (non-adaptive system).
From the viewpoint: of the end-users.
In the context of: end-users using the managed system on a mobile device.

From this objective, the following hypotheses were derived:

Null hypothesis 1, H_{10}: The user experience using our adaptive system is the same as the one obtained using a non-adaptive system.
Alternative hypothesis 1, H_{11}: The user experience using our adaptive system is greater than the one obtained using a non-adaptive system.

7.3.1. *Variables*

We identified two types of variables:

– **Dependent variables:** Variables that correspond to the outcomes of the experiment. In this work, user experience was the target of the study, which was measured in terms of pragmatic and hedonic attributes following the model of Hassenzahl [61]. Specifically, these attributes measure the *pragmatic manipulation*, the *hedonic stimulation*, the *hedonic identification*, and the *attraction*.
– **Independent variables:** Variables that affect the dependent variables and were intentionally varied during the experimentation. The managed system was identified as a independent variable that affects the dependent variable. This variable had two alternatives: (1) *adaptive system* (based on our self-adaptive software infrastructure) and (2) *non-adaptive system* (traditional system without adaptation capabilities).

7.3.2. *Experiment context*

The context in which the experiment was carried out is described below. We describe the subjects that participated in the experiment, the objects that were studied in the experiment, and the instruments that were chosen to carry out the experiment.

Participants. Fifteen people participated in this experiment (9 men and 6 women). Their ages ranged from 19 to 50. Most of them were computer science students (master or PhD) and daily mobile device users. The participants received no compensation for their participation.

Objects of study. The objects used in the experiment were two managed systems: one running over our self-adaptive software infrastructure with interaction obtrusiveness adaptation capabilities (as the one explained in Section 6) and another without adaptation capabilities. Both systems had the same user interface to manage notifications. They were running on an iPhone 4 (iOS 5.1). The interaction components available for delivering the notifications in this system were: vibration (*vibration* feature), loud and soft audio (*sound* features), voice notifications (*speech* feature), a badge icon (*icon* feature), an alert (*highlight* feature), a momentary banner on the screen (*momentary* feature), and a static banner on the locked screen (*quick view* feature). By means of these interaction components and combinations of them, we were capable of giving support to all of the obtrusiveness levels for the adaptive system. In the case of the non-adaptive system, the default mobile configuration was used to deliver all the notifications (i.e., sound with a dialog alert corresponding to the *completely-aware* level of attention).

On the server side, the target platform that was used in our experiment was the open source implementation of OSGi Equinox Release 4. To run the instance of Equinox, we used a host with an Intel Core i7 1.8 GHz processor and 4 GB RAM 1333 MHz with Mac OS X Lion and Java 1.6.0_29 installed.

Instrumentation. The instruments used to carry out the experiment were:

– **A case study:** We used a case study scenario that describes different services notifications of users' daily life. Specifically, we defined a case study for the services notifications of a routine day in the context of a university professor. The experiment was conducted using an adaptive version of the scenario for the adaptive system and a non-adaptive one for the traditional system. The scenario focused on the interaction notification mechanisms used in different contexts because the adaptation of attentional resources is a key factor in the user experience of services of this kind [62]. In the adaptive version, the different service notifications were adapted dynamically at different obtrusiveness levels depending on the user context. In the non-adaptive version, all the notifications were presented at the same obtrusiveness level using the same interaction mechanisms regardless of the user context. This allowed us to identify the relevance of the obtrusiveness adaptation in the user experience for the interactions. The adaptive version of the case study is described as follows:

The scenario focuses on the daily routine of a university professor named Matt. He lives in a smart home with pervasive services with his wife and his son. Every day, he gets up at 7 a.m. and takes a shower. On this particular day, while he was in the bathroom, the washing machine notified him that it was fully loaded and ready to start. Because he had the mobile device in his room, the notification was presented at a *slightly appreciable* level of attention since he was not going to be aware of it at that moment. Then, during breakfast, the healthcare service reminded him to take his vitamins. Since he was alone with the mobile device on the table, the notification appeared at the *aware* level of attention by voice notifications. When he was leaving home to go to work, the weather service suggested that he takes an umbrella. Since he had the mobile device at hand, the notification was presented at the *slightly* level of attention.

While he was driving to work with a workmate, the agenda service reminded him of a meeting to attend at work. Because he was driving with someone, the notification was presented at a *slightly appreciable* level of attention. At work, Matt was having coffee with his workmates and the facebook service notified him of a post on his timeline. The notification appeared at a *slightly* level of attention in order not to disturb the conversation. However, this service has less priority for the user and a momentary banner with vibration were used. During the meeting, the agenda service reminded Matt about a deadline that was approaching. Because he was in the meeting, the notification appeared in an *invisible* manner so as not to interrupt the meeting. After the meeting, he

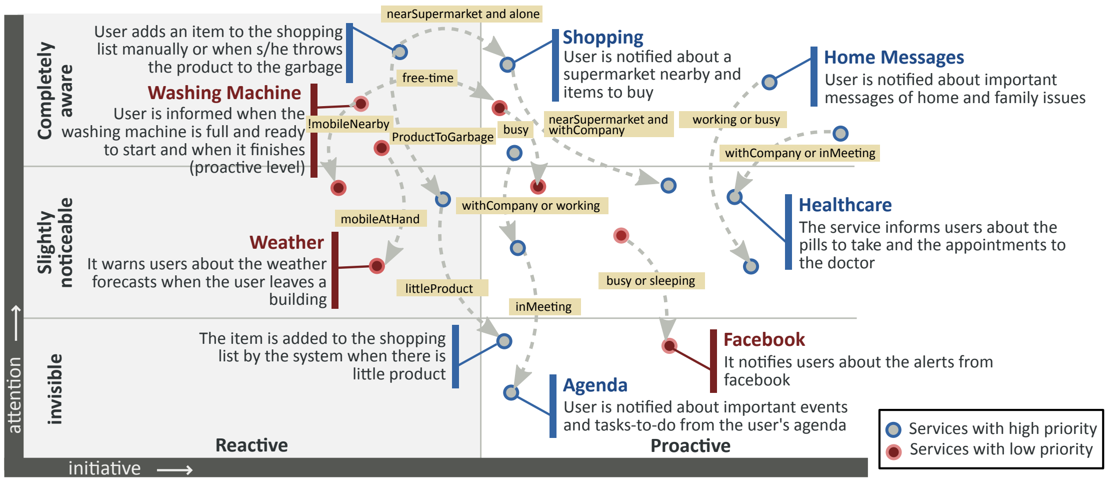

Completely aware
Slightly noticeable
invisible
attention
initiative
Reactive
Proactive
User adds an item to the shopping list manually or when s/he throws the product to the garbage
Washing Machine
User is informed when the washing machine is full and ready to start and when it finishes (proactive level)
Weather
It warns users about the weather forecasts when the user leaves a building
The item is added to the shopping list by the system when there is little product
Shopping
User is notified about a supermarket nearby and items to buy
Home Messages
User is notified about important messages of home and family issues
Healthcare
The service informs users about the pills to take and the appointments to the doctor
Facebook
It notifies users about the alerts from facebook
Agenda
User is notified about important events and tasks-to-do from the user's agenda
nearSupermarket and alone
free-time
!mobileNearby
ProductToGarbage
busy
nearSupermarket and withCompany
working or busy
withCompany or inMeeting
mobileAtHand
withCompany or working
littleProduct
inMeeting
busy or sleeping
Services with high priority
Services with low priority

Fig. 9. Unobtrusive adaptation spaced defined for the services of the case study.

was having lunch with his workmates, and the healthcare service reminded him that he had to take his pills. This notification was presented in a *slightly appreciable* manner due to the privacy of the message. In the afternoon, Matt was giving a course at the university and two notifications were made that required different levels of attention: one from the facebook service in an *invisible* manner (using an icon) and another one from the home messages service in a *slightly appreciable* manner (using a static banner with soft sound).

When he was going back home, he was near a supermarket and the mobile device notified him about an item to buy. Because he was alone and there was an urgent item on the shopping list, the notification was completely *aware* using the voice notification of the car. When Matt arrived home, he was watching TV and the washing machine reminded him at the highest level of attention to get the laundry out because he was not engaged in an important task. Then, Matt went running, listening to music with headphones. A notification from the agenda about a deadline approaching increased its attention level because Matt had not been aware of the notification that morning. This time, the notification was presented at the *aware* level of attention. At the end of the day, Matt was sleeping and the facebook service suggested to him a friend request Since he was sleeping, the notification was *invisible* in order not to disturb him (using an icon).

The goal of this adaptive version of the scenario was to show users the capabilities of AdaptIO by emphasizing the following points:

∗ The notification interaction is adjusted according to the user's context in terms of obtrusiveness (e.g., when the user is in a meeting, the notification is presented silently).
∗ In the same context, different services can be presented in different obtrusiveness levels based on user preferences about services (e.g., when the user is giving a course, the facebook and home messages are presented at a different obtrusiveness level).
∗ Different unobtrusive adaptation spaces can be designed with a different mapping to interaction resources in order to differentiate priorities between services (e.g., the healthcare and the facebook service in the *slightly* obtrusiveness level use different interaction resources because they have different priorities and they are modeled in different unobtrusive adaptation spaces).

Figure 9 shows the design of the services of the case study in the unobtrusive adaptation space for Matt according to the different contexts, priorities of services and obtrusiveness demands. In order to represent the priorities between services we have classified services in two unobtrusive adaptation spaces: one with high priority and another with low priority. In this way, interactions of services with less priority are more subtle and less obtrusive. For

Table 3
Interaction features for each obtrusiveness level of the unobtrusive adaptation spaces for the high and low priority.

Obtrusiveness Level	Active Interaction Features	Concrete Interaction Components
High Priority		
(proactive, aware)	visual, text, property, momentary, auditory, speech	momentary banner, speech
(proactive, slightly noticeable)	visual, text, property, quick view, auditory, sound, soft sound	static banner, soft sound
(proactive, invisible)	visual, image, property, iconic, haptic, vibration	badge icon, vibration
(reactive, aware)	visual, text, property, highlight, auditory, loud sound, radio, pointing	alert, loud sound, QR Code
(reactive, slightly noticeable)	visual, text, property, quick view, haptic, vibration, radio, touching	static banner, vibration, RFID/NFC
(reactive, invisible)	radio, scanning	bluetooth or GPS
Low Priority		
(proactive, aware)	text, property, highlight, auditory, loud sound	alert, loud sound
(proactive, slightly noticeable)	visual, text, property, momentary, haptic, vibration	momentary banner, vibration
(proactive, invisible)	visual, image, property, iconic,	badge icon
(reactive, aware)	text, property, highlight, auditory, loud sound, radio, pointing	alert, loud sound, QR Code
(reactive, slightly noticeable)	visual, text, property, quick view, auditory, soft sound, radio, touching	static banner, soft sound, RFID/NFC
(reactive, invisible)	radio, scanning	bluetooth or GPS

the sake of brevity, we do not have shown in the Fig. the opposite transitions with the corresponding opposite context (e.g., *withCompany* vs *alone*) and services with different priorities are represented in the same unobtrusive adaptation space but with different color.
Then, in order to support the behavior of the services in the different obtrusiveness levels, we have linked each obtrusiveness level with the appropriate interaction resources. Specifically, we have used the interaction features of Fig. 2. Table 3 shows the interaction configurations (active features) for each obtrusiveness level of both unobtrusive adaptation spaces and their corresponding concrete interaction components for the devices of the case study. With regard to the interaction of the high priority unobtrusive adaptation space, in the aware levels, the interaction is mainly auditory with visual support. In the slightly noticeable levels the interaction is visual (static banner) with little support of soft sound or vibration and finally in the invisible levels the interaction is minimal using an icon and vibration. In the reactive levels, the radio medium is used to provide identification capabilities by means of using QR Codes (pointing), RFID (touching) or scanning the environment using bluetooth or GPS. Regarding the low priority space, an alert dialog with sound support are used in the aware levels (mainly visual support); a banner with soft sound or vibration are used in the slightly levels; and a badge icon is used in the invisible levels. In the case of the radio mechanisms the classification is the same as the high priority space. Figure 10 shows some screenshots of the interaction provided for some services in its corresponding obtrusiveness level according to the context of the case study. It is worth noticing that the obtrusiveness of services for the adaptive version was adjusted for a professor's profile and his needs. For each user, this design should be adjusted based on his/her needs following a user-centered design process, such

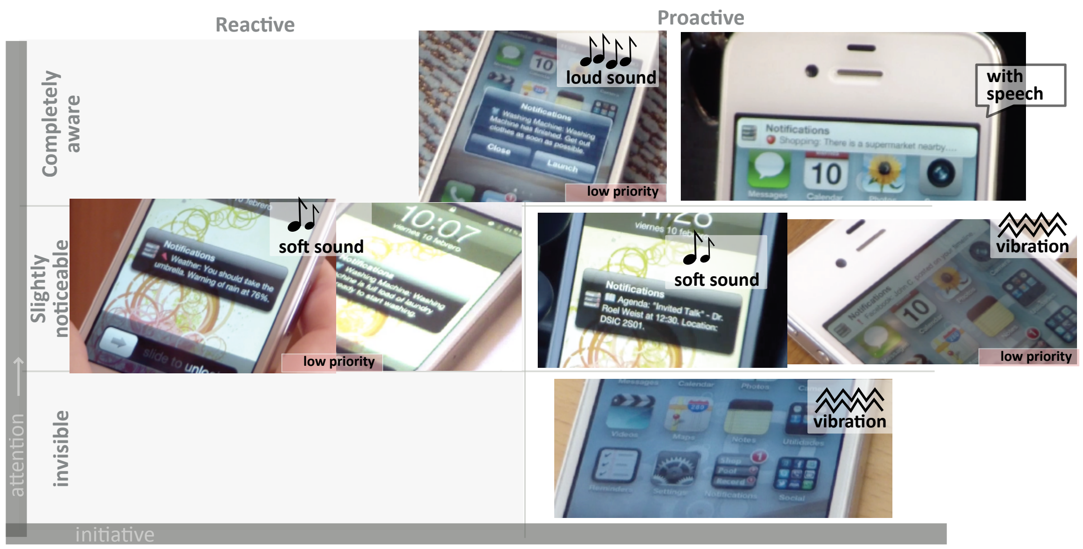
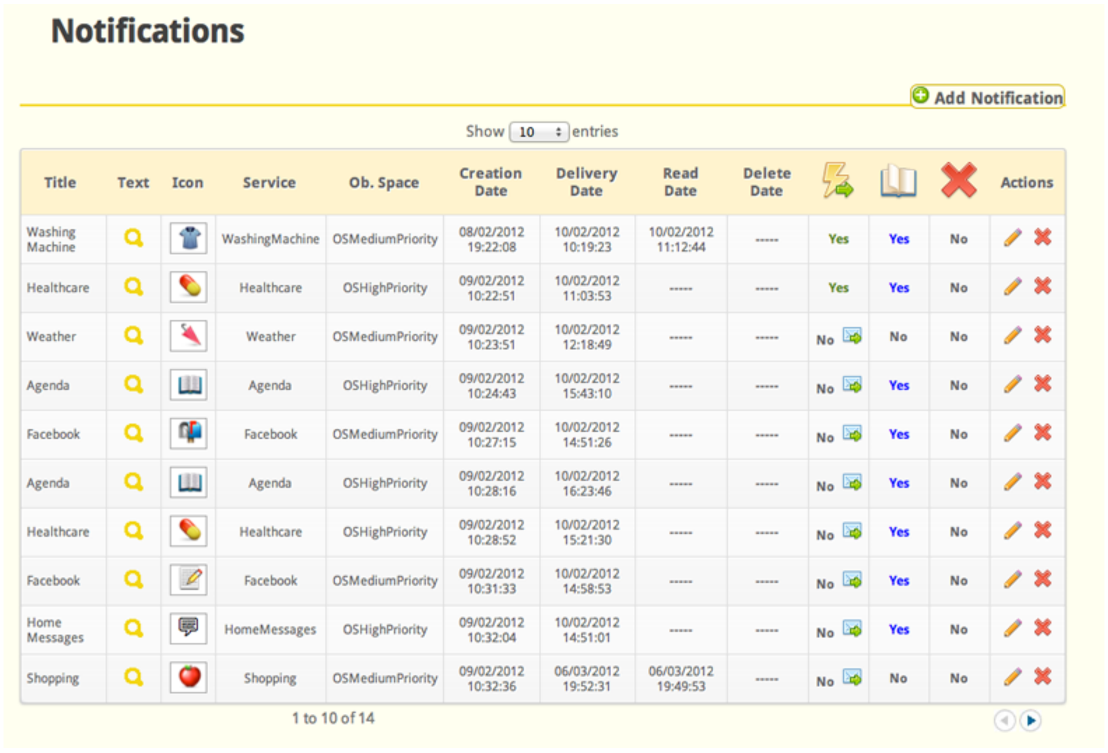


Fig. 10. Screenshots of the interaction provided by some services in different obtrusiveness levels.

as the one presented in [7]. Regarding the non-adaptive version of the case study, all the notifications were presented in the same order but in the *aware* level of attention regardless the user context.

- **Guidelines:** The description of the activities that the subjects should carry out in the experiment in order to adopt Matt's role and perform the scenario of the case study. These guidelines guided the participants to perform the activities of the case study and simulate the different user contexts during the experiment (e.g., having lunch, in a meeting, etc.). In each context, users received the appropriate notifications according to the case study description. An example of guideline is "now take a coffee", "simulate to go to work", or "now you are in a meeting". These guidelines helped us to take more enclosed the experiment and avoid test errors. These guidelines correspond to the tasks that the users have to do in the experiment for each system scenario.
- **A video prototype:** A conceptual video prototype was created to show the key concepts of the interaction obtrusiveness adaptation and the different user situations and activities that the users should perform. A video prototype makes users to be familiar with the adaptations and provides a quick exploration of the user experience by using our system. This video also helped to clarify the guidelines to be followed by users[19].
- **A questionnaire:** To measure the UX, we followed one of the most influential models proposed by Hassenzahl [63]. According to this model, each interactive system has a pragmatic (related to usability) and hedonic (related to users' self) quality that contributes to the UX. Based on this model, we use the AttrakDiff 2 questionnaire [63] to measure UX. The questionnaire consists of twenty-one 7-point items with bipolar verbal anchors, ranging from -3 to 3, where zero represents the neutral value between the two anchors of the scale. It is composed of four main constructs: *Pragmatic Quality* (PQ), which is related to traditional usability issues (e.g., effectiveness, efficiency, learnability, etc.); *Hedonic Quality Stimulation* (HQ-S), which is related to the personal growth of the user and the need to improve personal skills; *Hedonic Quality Identification* (HQ-I), which is focused on the human need to be perceived by others in a particular way; and *Attraction* (ATT), which is about the global appeal of the system. The questionnaire also included questions about personal information and background.
- **A notification management system:** To simulate the different services and deliver the noti-

[19]The video can found at: https://tatami.dsic.upv.es/adaptio/dissemination.php



Title	Text	Icon	Service	Ob. Space	Creation Date	Delivery Date	Read Date	Delete Date				Actions
Washing Machine	🔍		WashingMachine	OSMediumPriority	08/02/2012 19:22:08	10/02/2012 10:19:23	10/02/2012 11:12:44	-----	Yes	Yes	No	✎ ✖
Healthcare	🔍		Healthcare	OSHighPriority	09/02/2012 10:22:51	10/02/2012 11:03:53	-----	-----	Yes	Yes	No	✎ ✖
Weather	🔍		Weather	OSMediumPriority	09/02/2012 10:23:51	10/02/2012 12:18:49	-----	-----	No	No	No	✎ ✖
Agenda	🔍		Agenda	OSHighPriority	09/02/2012 10:24:43	10/02/2012 15:43:10	-----	-----	No	Yes	No	✎ ✖
Facebook	🔍		Facebook	OSMediumPriority	09/02/2012 10:27:15	10/02/2012 14:51:26	-----	-----	No	Yes	No	✎ ✖
Agenda	🔍		Agenda	OSHighPriority	09/02/2012 10:28:16	10/02/2012 16:23:46	-----	-----	No	Yes	No	✎ ✖
Healthcare	🔍		Healthcare	OSHighPriority	09/02/2012 10:28:52	10/02/2012 15:21:30	-----	-----	No	Yes	No	✎ ✖
Facebook	🔍		Facebook	OSMediumPriority	09/02/2012 10:31:33	10/02/2012 14:58:53	-----	-----	No	Yes	No	✎ ✖
Home Messages	🔍		HomeMessages	OSHighPriority	09/02/2012 10:32:04	10/02/2012 14:51:01	-----	-----	No	Yes	No	✎ ✖
Shopping	🔍		Shopping	OSMediumPriority	09/02/2012 10:32:36	06/03/2012 19:52:31	06/03/2012 19:49:53	-----	No	No	No	✎ ✖

1 to 10 of 14

Fig. 11. Notification management system.

fications to subjects while they were in the different user contexts, we implemented a notification management system (see Fig. 11). This system supports the introduction of notifications for the different services and their delivery to the managed systems. With this system, we were able to simulate the notifications from pervasive/mobile services in order to deliver them in the timing of the case study. Note that this system is transparent to users. An operator provides the appropriate notifications according to the user context using another device. In this way, the user is immersed in an environment that behaves like a working system, but it is much easier to test in an experimental context.

7.3.3. *Experimental design and procedure*

We followed a *within-subjects design* where all subjects were exposed to every treatment/system (adaptive and non-adaptive). In order to minimize the effect of the order in which the subjects applied the systems, the order was assigned randomly to each subject. Also, we had the same number of subjects starting with the first system as with the second in order to have a balanced design and to minimize order effects. In this way, we minimized the threat of learning from previous experience. As all the subjects had experience in the use of traditional notification systems (notification systems in their mobile phone), influences between systems were minimal.

Procedure. The study was initiated with a presentation in which general information and instructions were given. Next, we asked users some questions to capture the user's background such as their experience in the use of smart phones and notification systems. Afterwards, the guidelines were given to the subjects and the video was shown to familiarize users with the case study. Then, users started to follow the guidelines, adopting Matt's role, and performing the activities described in the case study (to simulate the different user contexts). Meanwhile, notifications were presented to the users in the different situations according to the case study description (we sent notifications by means of using the notification management system to send the notification in the adequate moment). Half of the subjects started the study with the non-adaptive system and the other half started with the adaptive system. The users performed the same tasks for both systems following the guidelines. For each system, users filled in the questionnaire to capture the user experience perception (their emotional feeling) about the used system. Specifically, they rated each system based on the notifications (interfaces) presentation in the different user contexts. Between the evaluation of systems, we gave a break to minimize the interference between systems. The whole study took about 30 minutes per user with a short break of 5 minutes in the middle.

7.3.4. *Validity evaluation*

The various threats that could affect the results of this experiment and the measures that we took were the following:

- **Conclusion validity:** This validity is concerned with the relationship between the treatment and the outcome. Our experiment was threatened by the *random heterogeneity of subjects*. This threat appears when some users within a user group have more experience than others. In the context of our experiment, this threat was minimized by selecting users with similar background in the use of smartphones and notification systems and introducing personal questions in the questionnaire. Also, our experiment was threatened by the *reliability of measures* threat since the validity of an experiment is highly dependent on this. In general, objective measures, that can be repeated with the same outcome, are more reliable than subjective measures. In order to reduce this threat, we used the AttrakDiff 2 questionnaire that measures hedonic stimulation and identity and pragmatic qualities of software products.
- **Internal validity:** This type of validity concern is related to the influences that can affect the factors with respect to causality, without the researcher's knowledge. Our evaluation had the *maturation* threat: the effect that users react differently as time passes (because of boredom or tiredness). We solved this threat by giving a five-minutes break between the evaluation of the two systems. Another internal validity threat that our evaluation had was *instrumentation*: even though tasks and questionnaires are the same for all subjects, a wrong interpretation of the task may affect the results. This threat was minimized by showing a video prototype of the activities to be performed by users before the experiment.
- **Construct validity:** Threats to construct validity refer to the extent to which the experiment setting actually reflects the construct under study. Our experiment was threatened by the *hypothesis guessing* threat: when people might try to figure out what the purpose and intended result of the experiment is and they are likely to base their behavior on their guesses. We minimized this threat by hiding the goal of the experiment.

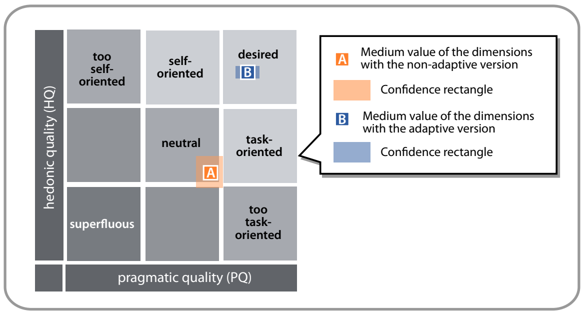


Fig. 12. Average values of the dimensions PQ and HQ and the respective confidence rectangles of both versions.

7.3.5. *Analysis and interpretation of results*

This section presents the analysis and interpretation of results related the user experience. Figure 12 represents the values of hedonic quality on the vertical axis (bottom = low value) and pragmatic quality on the horizontal axis (left = a low value) and the respective confidence rectangles for both versions. A small confidence rectangle is an advantage because it means that the investigation results are more reliable and less coincidental. As Fig. shows, in the case of the non-adaptive version of the system, it was rated as "neutral". The pragmatic confidence interval overlaps into the neighboring character zone. The user is assisted by the system, however the value of pragmatic quality only reaches the average values. Consequently, there is room for improvement in terms of usability. Regarding the adaptive version, users rated it as "desired". Pragmatic quality is clearly the classification, which means that the adaptive version assists its users optimally. In terms of hedonic quality the character classification applies positively which means that the user identifies with this version and is motivated and stimulated by it. In both cases, the confidence rectangle is small, meaning that the users are at one in their evaluation and in their ratings of both dimensions.

Figure 13 illustrates the mean values of the UX dimensions for both systems[20]. As mentioned earlier each answer gets a value from -3 to 3, with zero as the neutral value between the anchors of the question.

[20]Complete dataset of results at https://dl.dropboxusercontent.com/u/14910519/UX_RawData.sav

Table 4

Results of the paired t-test for each pair of measures

	Diff. between means	Std. deviation	Std. error	Confidence interval 95 % for the diff. Lower	Bound	t	Sig.
Pair PQ	-,98	,93	,24	-1,49	-,46	-4,07	,001
Pair HQ-I	-1,86	1,15	,29	-2,50	-1,22	-6,28	,000
Pair HQ-S	-3,28	1,06	,27	-3,87	-2,69	-11, 99	,000
Pair ATT	-2,67	,99	,25	-3,22	-2,12	-10,43	,000

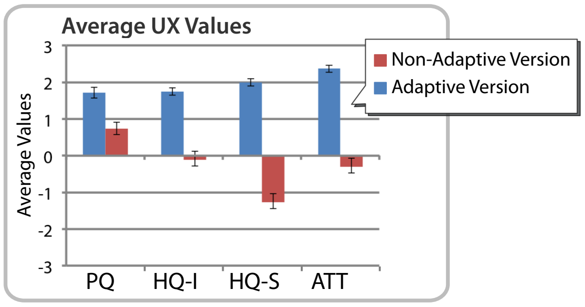


Fig. 13. Mean UX values of the four dimensions for both systems.

According to the results, we observed considerable deviation between the instances in HQ-S and ATT. In HQ-S, the mean scores for the non-adaptive system were -1.27 and 2 for the adaptive one. In ATT, the mean scores for the non-adaptive system were -0.30 and 2.37 for the adaptive one. These values indicate that the users perceived a huge difference in the hedonic quality (stimulation) and overall appeal (attraction) between both versions, considering our system better in these aspects. Stimulation is enhanced by presenting things in a novel way or by a new interaction style. In the case of the HQ-S for the non-adaptive version, the system is located in the below-average region meaning that the system does not have a stimulating effect on users. For the PQ, we observed less difference between the mean values. Even so, the PQ results were higher in the adaptive system (1.72), indicating that the users considered this system more usable and enables them to achieve better their aims. In the case of the non-adaptive version, the PQ is located in the average region (0.74), meeting ordinary standards. Results also indicate that HQ-I does not primarily affect any of the systems, since it is intended for evaluation of products rather than software. However, the HQ-I of the adaptive version is located in the above-average region (1.75), while for the non-adaptive version is located in the average region.

In order to study the comparison of independent variables in depth, we performed a statistical analysis of results called *paired t-test*. This test is a parametric statistical test used to compare two sets of scores that come from the same participants. In our experiment, it is used to evaluate whether there was a difference in overall user experience under two versions of systems: non-adaptive versus adaptive. Statistical analysis has been carried out using the IBM SPSS Statistics V20 at a confidence level of 95% (α=0.05) [64]. When the critical level (the significance) is higher than 0.05, we cannot reject the null hypothesis because there is high probability that the differences of values happened due to randomness.

First, we verified whether or not the dependent variables followed a normal distribution. To achieve this, we applied a one-sample Kolmogorov-Smirnov (K-S) test. The results of the K-S test showed that all the dependent variables where normally distributed since all the values were greater than 0.05. Then, we performed the paired t-test. Table 4 shows the results obtained. The analysis showed that difference in mean scores non-adaptive and adaptive was significant in all the dimensions: PQ (meanDiff = -,98, t = -4,07, Sig = .001), HQ-I (meanDiff = -1,86, t = -6,28, Sig = .000), HQ-S (meanDiff = -3,28, t = -11,99, Sig = .000), ATT (meanDiff = -2,67, t = -10,43, Sig = .000). As the mean scores for all the dimensions of the adaptive version are higher than the mean scores of the non-adaptive version and the difference of means is statistically significant between both systems, we can reject the null hypothesis H_{10} (the user experience using our adaptive system is the same as the one obtained using a non-adaptive system) with a two-tailed test at the 0.05 level. Based on this test, we have given evidence that the kind of system influences the user experience. Specifically, the user experience with the obtrusiveness adaptations (adaptive version) is significantly better than without adaptations (non-adaptive version); thus, the alterna-

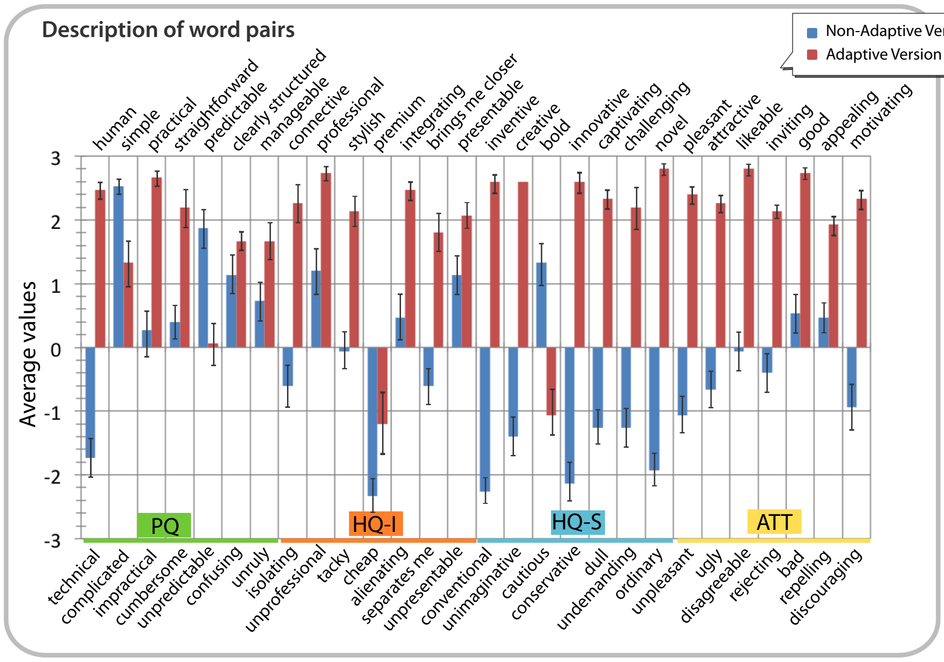


Fig. 14. Mean values of the word pairs for both versions.

tive hypothesis H_{11} is fulfilled: *the user experience using our adaptive system is greater than the one obtained using a non-adaptive system.*

With regard to the mean values of the word pairs, in our research, there were some that were of particular interest. Figure 14 shows the detailed mean values of these pairs. The results indicated that the adaptive system is significantly more *human* and *cautious* than the non-adaptive one. This is because the adaptive system attunes notifications to the user attention and context behaving like a human and avoiding interrupting the user. The users rated the adaptive system as being a bit more *complicated* since it had more configuration features to take into account compared to the non-adaptive system, which only had basic configuration features. Also, the users considered our system as being a little more *unpredictable* compared to the non-adaptive one, since at first they were not sure how they would receive the notifications. Despite these detected issues, the adaptive version was considered more *likeable*, *pleasant*, and *attractive* for users because they considered that with this version they can have a smoother everyday experience. Some users said they might become frustrated with the everyday interruptions of service notifications. Thus, the results give first insights that the user experience is better with our obtrusiveness adaptations. It is worth noticing that this experiment (small-scale study) is a first attempt to explore if the obtrusiveness of interactions affects the user experience. In order to generalize the results a large-scale exploration should be carried out.

8. Discussion

During the application of our proposal, we observed some benefits and limitations. Our proposed approach has two main aspects:

- **Reuse of the design knowledge to achieve adaptation.** In order to guide the adaptation of the interaction, we leverage models at runtime without modification (i.e., we keep the same model representation at runtime that we use at design time). The models can provide us with a richer semantic base for runtime decision-making related to system adaptation since all the information analyzed at design time is also available at runtime.
- **Support system evolution**. The fast changing nature of user preferences and the technological heterogeneity of ubiquitous devices suggest that systems in this area must be designed to evolve. As the provided software infrastructure directly interprets the design models at runtime, this facilitates considerably the evolution of the system at

runtime: as soon as the models are changed, the evolutions are applied by the infrastructure. Thus, the models become the primary means to understand, interact with, and modify the service interactions and their obtrusiveness level.

Although the application of the approach achieved satisfactory results, some complaints that are commonly found in adaptive systems were present in our proposal to some extent. These are the extent to which users have control over adaptations, the handling of failures due to imperfect context sensing and a particular demand on handling special messages. More detail about the issues detected is provided below.

- **Intelligibility.** During the experiments, we found that *intelligibility* can become an issue that affects user satisfaction because the adaptation is transparent to users and automatically performed. This causes loss of control over the system and the feeling that the system is doing something “behind our backs". Context-aware applications should be *intelligible* (also called transparent, comprehensible, scrutable), capable of generating explanations for their behavior [65]. Thus, our system should explain its decisions to the user in some way that is not intrusive for him/her. A way to incorporate intelligibility in our approach is by means of generating explanations of the system behavior [66]. These explanations usually answer the questions (*Input, Output, What, Why, Why Not, How To, What If, Certainty, Description*) generated from the knowledge models.
- **Handling failures and conflicts.** The designed adaptations are exposed to possible failures due to imperfect context sensing or failures in the mobile device. As Bellotti and Edwards pointed out [65], a context-aware system cannot be expected to understand the entire user context and therefore must adjust to its own limitations. Furthermore, this situation is aggravated when systems run in an unpredictable environment and they have to deal with high levels of uncertainty both in assessing what the context really is and what the appropriate reaction of the system should be. In this situations, the system should be able to handle these failures and self-heal accordingly at runtime. This can be achieved in our approach by means of defining reconfiguration rules to indicate the behavior of the system when a failure is detected. Also, probabilistic logic could be used for improving the quality of context information through multi-sensor fusion as well as for deriving higher-level probabilistic contexts [67]. Another way of dealing with complex situations that are difficult to resolve autonomously is involving humans in the adaptation process to help the system facing these conflicts difficult to solve by itself ("human in the loop") [68].
- **Handling special messages.** Our system currently works at per-service level by acting on user’s context changes. However, in the case of a notification system, the same service could provide different messages that could need different attention level based on the relevance of the message for the user. Message relevance has been shown as a primary factor that affects attention-aware interactions [69]. Our system could be able to work also at per-message level by introducing *temporal transitions* in the unobtrusive adaptation space. A temporal transition allows a service to temporarily transition to another obtrusiveness level when a special message is received (e.g., an urgent message), handle the service in that level, and then return back to the previous level. In order to support these temporal transitions, services should have associated metadata with certain messages, for instance, specifying that the message is urgent. Finally, the conditions of these temporal transitions should refer to the metadata associated with these special messages (e.g., message = urgent). However, this limitation is specific for notification services.

9. Conclusions and future work

In this work, we have used model-based adaptation techniques to self-adapt service interactions in terms of obtrusiveness. With the increase in the capabilities of mobile and pervasive devices, user attention becomes a bottleneck for the system. Therefore, it is important to be able to manage it in an effective manner. A self-adaptive software infrastructure has been defined in order to automatically adapt the interaction of the different services according to the momentary attentive state of users. High-level design models are exploited at runtime to drive the autonomic adaptation of interaction obtrusiveness. AdaptIO takes into account obtrusiveness aspects in order to adapt the way in which service interactions are provided in each situation to avoid overwhelming the user.

The mechanisms provided for adapting interaction obtrusiveness at runtime help services to be considerate with the user according to his/her context. Also, the use of design models at runtime offers new opportunities for adaptation capabilities without increasing development costs. This is accomplished by means of a planned reutilization of the efforts invested at design time. In addition, the definition of the infrastructure at the modeling level allows the system to be sustainable since it can support its evolution to new technologies.

In our approach, the design effort dedicated to analyze the system is reused at runtime: the design of the system constitutes its adaptation behavior at runtime. Also, we facilitate the systematic reuse of contextual knowledge (user situation rules) and adaptation rules (obtrusiveness levels and interaction resources) between services. Another advantage is dealing with user attention as a separate concern from other aspects. In this way, a given service can be presented to the user in a complete different manner by only changing the obtrusiveness specification, without altering its description. Moreover, we provide tools for specifying the obtrusiveness levels of each service in a graphical way, facilitating the adaptation description and the integration of the obtrusiveness with the service interactions.

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. Thus, for future work, we plan to (1) offer users the possibility to check the adaptations in order to reduce the uncertainty and lack of control over the adaptive systems, (2) introduce mechanisms to discover new environments and the services that are in these new environments and adapt the interaction with these services in a considerate manner, and (3) extend the infrastructure to take into account special messages by working also at per-message level.

Acknowledgements

This work has been developed with the support of MINECO under the project SMART-ADAPT TIN2013-42981-P, and co-financed by the *Generalitat Valenciana* under the postdoctoral fellowship APOSTD/2016/042.

References

[1] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, G. Papadopoulos, A development framework and methodology for self-adapting applications in ubiquitous computing environments, Journal of Systems and Software 85 (12) (2012) 2840 – 2859.

[2] D. J. Patterson, C. Baker, X. Ding, S. J. Kaufman, K. Liu, A. Zaldivar, Online everywhere: evolving mobile instant messaging practices, in: Proceedings of the 10th international conference on Ubiquitous computing, UbiComp '08, ACM, New York, NY, USA, 2008, pp. 64–73.

[3] H. Chen, J. P. Black, A quantitative approach to non-intrusive computing, in: Mobiquitous '08: Proceedings of the 5th Annual International Conference on Mobile and Ubiquitous Systems, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2008, pp. 1–10.

[4] M. Tedre, What should be automated?: The fundamental question underlying human-centered computing, in: Proceedings of the 1st ACM international workshop on Human-centered multimedia, HCM '06, ACM, New York, NY, USA, 2006, pp. 19–24.

[5] M. Weiser, J. S. Brown, The coming age of calm technolgy, in: P. J. Denning, R. M. Metcalfe (Eds.), Beyond calculation, Copernicus, New York, NY, USA, 1997, pp. 75–85.

[6] W. W. Gibbs, Considerate computing, Scientific American 292 (1) (2005) 54–61.

[7] M. Gil, P. Giner, V. Pelechano, Personalization for unobtrusive service interaction, Personal Ubiquitous Comput. 16 (5) (2012) 543–561.

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

[9] A. Campbell, T. Choudhury, From smart to cognitive phones, Pervasive Computing, IEEE 11 (3) (2012) 7 –11.

[10] A. Ferscha, 20 years past weiser: What's next?, Pervasive Computing, IEEE 11 (1) (2012) 52 –61.

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

[12] S. Rosenthal, A. K. Dey, M. Veloso, Using decision-theoretic experience sampling to build personalized mobile phone interruption models, in: Proceedings of the 9th international conference on Pervasive computing, Pervasive 2011, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 170–187.

[13] D. Warnock, M. McGee-Lennon, S. Brewster, The Role of Modality in Notification Performance, Human-Computer Interaction – INTERACT 2011 6947 (Chapter 43) (2011) 572–588.

[14] E. Horvitz, P. Koch, R. Sarin, J. Apacible, M. Subramani, Bayesphone: precomputation of context-sensitive policies for inquiry and action in mobile devices, in: Proceedings of the 10th international conference on User Modeling, UM 2005, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 251–260.

[15] M. Valtonen, A.-M. Vainio, J. Vanhala, Proactive and adaptive fuzzy profile control for mobile phones, in: IEEE International Conference on Pervasive Computing and Communications, 2009. PerCom 2009., 2009, pp. 1 –3.

[16] E. Serral, Automating routine tasks in smart environments: A context-aware model-driven approach, Ph.D. thesis, Universitat Politecnica de Valencia (2011).

[17] J. Kephart, D. Chess, The vision of autonomic computing, Computer 36 (1) (2003) 41 – 50.

[18] D. Siewiorek, A. Smailagic, J. Furukawa, A. Krause, N. Moraveji, K. Reiger, J. Shaffer, F. L. Wong, Sensay: A context-aware mobile phone, in: Proceedings of the 7th IEEE International Symposium on Wearable Computers, ISWC '03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 248–.

[19] P. Korpipaa, E.-J. Malm, T. Rantakokko, V. Kyllonen, J. Kela, J. Mantyjarvi, J. Hakkila, I. Kansala, Customizing user interaction in smart phones, IEEE Pervasive Computing 5 (3) (2006) 82 –90.

[20] L. Barkhuus, A. Dey, Is context-aware computing taking control away from the user? three levels of interactivity examined, in: UbiComp 2003: Ubiquitous Computing, Vol. 2864 of Lecture Notes in Computer Science, 2003, pp. 149–156.

[21] C. Evers, R. Kniewel, K. Geihs, L. Schmidt, The user in the loop: Enabling user participation for self-adaptive applications, Future Generation Computer Systems 34 (0) (2014) 110–123.

[22] J. Floch, C. Frà, R. Fricke, K. Geihs, M. Wagner, J. Lorenzo, E. Soladana, S. Mehlhase, N. Paspallis, H. Rahnama, P. A. Ruiz, U. Scholz, Playing MUSIC - Building context-aware and self-adaptive mobile applications, Software - Practice and Experience 43 (3) (2013) 359–388.

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

[24] T. Clerckx, C. Vandervelpen, K. Coninx, Task-based design and runtime support for multimodal user interface distribution, in: Engineering Interactive Systems, Vol. 4940 of Lecture Notes in Computer Science, 2008, pp. 89–105.

[25] M. Aleksy, T. Butter, M. Schader, Context-aware loading for mobile applications, in: Network-Based Information Systems, Vol. 5186 of Lecture Notes in Computer Science, 2008, pp. 12–20.

[26] M. Blumendorf, G. Lehmann, S. Albayrak, Bridging models and systems at runtime to build adaptive user interfaces, in: Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems, EICS '10, ACM, New York, NY, USA, 2010, pp. 9–18.

[27] V. Motti, J. Vanderdonckt, A computational framework for context-aware adaptation of user interfaces, in: IEEE Seventh International Conference on Research Challenges in Information Science (RCIS), 2013, 2013, pp. 1–12. doi:10.1109/RCIS.2013.6577709.

[28] S. Ramchurn, B. Deitch, M. Thompson, D. De Roure, N. Jennings, M. Luck, Minimising intrusiveness in pervasive computing environments using multi-agent negotiation, in: Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on, 2004, pp. 364 – 371.

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

[30] B. Poppinga, W. Heuten, S. Boll, Sensor-Based Identification of Opportune Moments for Triggering Notifications, Pervasive Computing, IEEE 13 (1) (2014) 22–29.

[31] M. Pielot, R. de Oliveira, H. Kwak, N. Oliver, Didn't you see my message?: Predicting attentiveness to mobile instant messages, in: Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems, CHI '14, ACM, New York, NY, USA, 2014, pp. 3319–3328. doi:10.1145/2556288.2556973.

[32] J.-Y. Mao, K. Vredenburg, P. W. Smith, T. Carey, User-centered design methods in practice: a survey of the state of the art, in: Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research, CASCON '01, IBM Press, 2001, pp. 12–.

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

[34] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, D. Riboni, A survey of context modelling and reasoning techniques, Pervasive and Mobile Computing 6 (2) (2010) 161–180.

[35] E. Serral, P. Valderas, V. Pelechano, Towards the model-driven development of context-aware pervasive systems, Pervasive and Mobile Computing 6 (2) (2010) 254–280.

[36] W. Woensel, M. Gil, S. Casteleyn, E. Serral, V. Pelechano, Adapting the obtrusiveness of service interactions in dynamically discovered environments, in: K. Zheng, M. Li, H. Jiang (Eds.), Mobile and Ubiquitous Systems: Computing, Networking, and Services, Vol. 120 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer Berlin Heidelberg, 2013, pp. 250–262.

[37] M. de Sá, C. Duarte, L. Carriço, T. Reis, Designing Mobile Multimodal Applications, Information Science Reference, 2010, Ch. 5, pp. 106–136.

[38] K. Czarnecki, S. Helsen, U. Eisenecker, Staged configuration using feature models, in: Software Product Lines, Vol. 3154 of Lecture Notes in Computer Science, 2004, pp. 162–164.

[39] D. Benavides, P. Trinidad, A. Ruiz-Cortés, Automated reasoning on feature models, in: Proceedings of the 17th international conference on Advanced Information Systems Engineering, CAiSE'05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 491–503.

[40] Y. Bachvarova, B. van Dijk, A. Nijholt, Towards a unified knowledge-based approach to modality choice, in: Proc. Workshop on Multimodal Output Generation (MOG), 2007, pp. 5–15.

[41] N. O. Bernsen, Foundations of multimodal representations: a taxonomy of representational modalities, Interacting with Computers 6 (4) (1994) 347 – 371.

[42] Z. Obrenovic, J. Abascal, D. Starcevic, Universal accessibility as a multimodal design issue, Commun. ACM 50 (5) (2007) 83–88.

[43] S. Lemmelä, A. Vetek, K. Mäkelä, D. Trendafilov, Designing and evaluating multimodal interaction for mobile contexts, in: Proceedings of the 10th international conference on Multimodal interfaces, ICMI '08, ACM, New York, NY, USA, 2008, pp. 265–272.

[44] Model validation with fama framework, https://tatami.dsic.upv.es/moskitt4spl/features.php (Jun. 2016).

[45] Y. Cao, M. Theune, A. Nijholt, Modality effects on cognitive load and performance in high-load information presentation, in: Proceedings of the 14th international conference on Intelligent user interfaces, IUI '09, ACM, New York, NY, USA, 2009, pp. 335–344.

[46] R. E. Mayer, R. Moreno, Nine ways to reduce cognitive load in multimedia learning, EDUCATIONAL PSYCHOLOGIST 38 (2003) 43–52.

[47] E. Haapalainen, S. Kim, J. F. Forlizzi, A. K. Dey, Psycho-physiological measures for assessing cognitive load, in: Proceedings of the 12th ACM international conference on Ubiquitous computing, Ubicomp '10, ACM, New York, NY, USA, 2010, pp. 301–310. doi:10.1145/1864349.1864395.

[48] E. Rukzio, K. Leichtenstern, V. Callaghan, An experimental comparison of physical mobile interaction techniques: Touching, pointing and scanning, in: 8th International Conference on Ubiquitous Computing, UbiComp 2006, Orange County, California, 2006.

[49] M. Gil, Additional details of the adaptio infrastructure, https://tatami.dsic.upv.es/adaptio/system.php (Jun. 2016).

[50] Author, Tool support to design unobtrusive service interaction adaptation, https://tatami.dsic.upv.es/moskitt4spl/adaptio.php.

[51] M. Gil, Video of adaptio design suite, https://tatami.dsic.upv.es/adaptio/dissemination.php (Jun. 2016).

[52] R. Murch, Autonomic Computing, IBM Press, 2004.

[53] B. Cheng, R. De Lemos, H. Giese, P. Inverardi, Software engineering for self-adaptive systems.

[54] IBM, An architectural blueprint for autonomic computing, Tech. rep. (2006).

[55] M. Gil Pascual, Adapting Interaction Obtrusiveness: Making Ubiquitous Interactions Less Obnoxious. A Model Driven Engineering approach, Ph.D. thesis, Universitat Politècnica de València, Valencia (Spain) (jul 2013). doi:10.4995/Thesis/10251/31660.
URL `https://riunet.upv.es/handle/10251/31660`

[56] F. Chang, J. Ren, Validating system properties exhibited in execution traces, in: Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering, ASE '07, ACM, New York, NY, USA, 2007, pp. 517–520.

[57] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, R. Koschke, A systematic survey of program comprehension through dynamic analysis, Software Engineering, IEEE Transactions on 35 (5) (2009) 684–702. doi:10.1109/TSE.2009.28.

[58] S. Maoz, Using model-based traces as runtime models, Computer 42 (10) (2009) 28–36.

[59] D. J. Cook, S. K. Das, Review: Pervasive computing at scale: Transforming the state of the art, Pervasive and Mobile Computing 8 (1) (2012) 22–35.

[60] R. van Solingen, E. Berghout. Goal/question/measures [online] (1999).

[61] M. Hassenzahl, N. Tractingsky, User experience – a research agenda, Behaviour & Information Technology 25 (2) (2006) 91–97.

[62] M. H. Vastenburg, D. V. Keyson, H. Ridder, Considerate home notification systems: a field study of acceptability of notifications in the home, Personal Ubiquitous Comput. 12 (8) (2008) 555–566.

[63] M. Hassenzahl, The interplay of beauty, goodness, and usability in interactive products, Hum.-Comput. Interact. 19 (4) (2008) 319–349.

[64] J. Bruin, Statistical analyses using spss http://www.ats.ucla.edu/stat/spss/whatstat/whatstat.htm#1sampt (2011).

[65] V. Bellotti, K. Edwards, Intelligibility and accountability: human considerations in context-aware systems, Hum.-Comput. Interact. 16 (2) (2001) 193–212.

[66] B. Y. Lim, Improving trust in context-aware applications with intelligibility, in: Ubicomp '10 Adjunct: Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing - Adjunct, ACM Request Permissions, New York, New York, USA, 2010, p. 477.

[67] R. Fagin, J. Y. Halpern, N. Megiddo, A logic for reasoning about probabilities, Inf. Comput. 87 (1-2) (1990) 78–128.

[68] J. Cámara, G. Moreno, D. Garlan, Reasoning about Human Participation in Self-Adaptive Systems, in: SEAMS 2015, 2015, pp. 146–156.

[69] C. Roda, Human attention and its implications for human-computer interaction, Cambridge University Press, 2011.