Adapting Interaction Obtrusiveness: Making Ubiquitous Interactions Less Obnoxious

A Model Driven Engineering Approach

Miriam Gil Pascual

Supervisor:
Vicente Pelechano Ferragud

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Miriam Gil Pascual

Adapting Interaction Obtrusiveness: Making Ubiquitous Interactions Less Obnoxious

A Model Driven Engineering approach

PhD Thesis, July 2013

Centro de Investigación en Métodos de Producción de Software

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Miriam Gil Pascual

Adapting Interaction Obtrusiveness: Making Ubiquitous Interactions Less Obnoxious

A Model Driven Engineering approach

PhD Thesis, July 2013

Adapting Interaction Obtrusiveness: Making Ubiquitous Interactions Less Obnoxious. A Model Driven Engineering approach

This report was prepared by
Miriam Gil Pascual

Supervisor
Dr. Vicente Pelechano Ferragud

Centro de Investigación en Métodos de Producción de Software
Universitat Politècnica de València
Camí de Vera s/n, Edif. 1F
46022 - València, Spain

Tel: (+34) 963 877 007 (Ext. 83533)
Fax: (+34) 963 877 359
Web: http://www.pros.upv.es

Release date:	26-07-2013
Comments:	A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science at the Universitat Politècnica de València.
Rights:	©Miriam Gil, 2013

To my parents.

Acknowledgements

This Ph.D. has been a long but fulfilling journey. This journey would not have been possible to do without the invaluable support and encouragement of a several. Without these supporters, especially the select I'm about to mention, I may not have gotten to where I am today. I hope they have learnt half as much from me as I have learnt from them.

Firstly, my supervisor Dr. Vicente Pelechano, thank you for being such a great supervisor and also my friend. You are inspiring; your passion for research is contagious and motivating; your "sixth sense" to always be in the right path is unbelievable. You trusted in me from the very beginning, when you didn't even know me, and gave me the chance to begin this road. We've laughed together and you've also been kind when I've needed in the bad moments. Your feedback, support and advice have been a privilege and a fundamental part to get to the end of the trip. I cannot thank you enough. I am forever grateful. Thank you Pele!

Secondly, I would like to thank Joan Fons, the first person I met in the group. You convinced me to join our research group. From you, I heard the word "pervasive" for the first time. You have shown me the practical side of research. Your humor and friendly sarcasm have allowed me to laugh and lighten this road. You always say that I'm your "idol", and the truth is that you're also my "idol".

I would also like to thank Pau Giner, who introduced me in the research world. He was my mentor in my first years and assisted me with the writing of my first research papers. With him, I discussed the initial ideas which resulted in this thesis. His refreshing insights, motivation and working spirit helped me to grow up as a researcher. Although he hasn't finished the road with me, I want him to know how grateful I am to him.

My special gratitude to Ignacio Mansanet. We started working in the research group at the same time and you have been my friend, confidant, and support from that moment until now. We've laughed and cried together. You have known the answer to every question I've ever asked you regarding work and life. You have always been a tremendous help no matter the task or circumstance. Thanks Nacho for being persistent and encouraging, for believing in me, and for the many precious memories along the way. For all these reasons and many, many more, I'm eternally grateful.

Next, I'd like to thank Ismael Torres. You've also been my friend and confidant during this journey. Among other things, you have showed me the medicine to avoid stress and to be a little happier by swimming, cycling, and running. Thanks Isma, for teaching me that with hard work and persistence we can achieve all the goals that we propose, and for encouraging me to do things that I thought I wasn't capable to do.

Another stanch supporter and friend has been Bernardo Botella, who studied the degree with me. Having joined our research group in the final stages of my work, you have helped and supported me as much as possible. I always enjoy talking about everything with you. Your relaxed demeanor, patience, and warm and friendly heart have softened the last potholes of this road. Thank you.

Thanks Estefanía Serral and Pedro Valderas for your constructive discussions about my work as well as your effort joining me to write research papers to promote our work. Thanks to my "groupmates" Vicky and Manoli for creating such a good athmosfere in our group, and my "labmates" María, Clara, Mario, Salva, and Pablo for the good times we've spent together both in the lab and out of it. You all have an incredible sense of humor; you have kept things light and me smiling. Thanks Oscar for your great direction of our research center, and Ana for all the time you've saved me with your work. I also would like to thank all the members of the ProS research center, past and present, for making this journey a pleasant one. Thank you all for your friendship, support and interest in my work.

I have been fortunate to come across so many funny and good friends from Benitatxell, Valencia and elsewhere, without whom life would be bleak. This work has also been possible thanks to them, who have contributed in one way or another to a rich and less stressful life and helped me to enjoy the little free time this Ph.D. left me. Special thanks go to Vanesa for twenty eight years of sharing everything with me, and to Marta for being my conscience, my fan, phone comrade, etc.

Thanks to Javi, my little brother, for sharing with me your funny vision of the world and for keeping me always smiling. You occupy a special place in my soul. Even though the distance, I do care for you. Thanks to my grandparents, for your love above all. And a very special memory to my grandmother, although not having been able to see the end of this thesis, I'm sure she'd be very proud.

Finally, my parents Joaquin and Teresa; words cannot truly express how much I owe you both. You gave me life and have done nothing but support me throughout it. Both have instilled many admirable qualities in me and have given me a good foundation with which to meet life. Thank you so much for your unstinting love, trust, help and encouragement. This one's for you!

Miriam

Agraïments

Aquesta tesi doctoral ha estat un viatge llarg, però satisfactori. Aquest viatge no hauria estat possible fer-lo sense l'inestimable suport i ànims de moltes persones. Sense aquestes persones, i en especial d'unes quantes que en parlaré, no haguera pogut arribar on sóc avuí. Sols espere que elles hagen après la meitat de mi del que jo he après d'elles.

En primer lloc, al meu director de tesi Dr. Vicente Pelechano, gràcies per ser un excel·lent director i també el meu amic. Eres una inspiració per als demés; la teua passió per la investigació és contagiosa i motivadora; el teu "sisé sentit" per a anar sempre pel camí correcte és increïble. Vas confiar en mi des del primer moment, quan ni tan sols em coneixies, i em vas donar l'oportunitat d'iniciar aquest camí. Hem rigut junts, compartit històries i viatges junts, i també has estat comprensiu quan ho he necessitat en els mals moments. Els teus consells, dedicació, disponibilitat i orientació han sigut un privilegi i una part fonamental per a arribar al final del viatge. No puc agrair-te prou. T'estaré eternament agraïda. Gràcies Pele!

En segon lloc, m'agradaria donar les gràcies a Joan Fons, la primera persona que vaig conèixer al grup. Tu em vas convèncer per a unir-me al nostre grup de recerca. De tu vaig sentir la paraula "pervasiu" per primera vegada. Tu m'has mostrat el costat pràctic de la investigació. El teu humor i sarcasme simpàtic m'han fet riure i alleugerir aquest camí. Sempre em dius que sóc la teua “ídola”, i la veritat és que tu també ets el meu “ídol”.

També m’agradaria donar les gràcies a Pau Giner, qui em va introduir al món de la recerca. Ell va ser el meu mentor en els primers anys i em va ajudar amb la redacció dels primers treballs de recerca. Amb ell, vaig parlar de les idees inicials que han donat lloc a aquesta tesi. Les seves idees fresques, la seva motivació i el seu esperit de treball em van ajudar a créixer com a investigadora. Tot i que no ha acabat el viatge amb mi, vull que sàpiga que li estic molt agraïda.

Un agraïment molt especial a Ignacio Mansanet, pel teu entusiasme en aquesta tesi en tot moment. Els dos començàrem a treballar en el grup de recerca al mateix temps, i has estat el meu amic, confident i suport des d’aquell moment fins ara. Hem rigut i plorat junts. Has tingut resposta a cada pregunta que t’he fet ja fora sobre el treball o la vida. Sempre has estat una enorme ajuda sense importar la tasca o circumstància. Gràcies Nacho, per ser persistent i encoratjador, per creure en mi, i per els molts preciosos moments que hem passat al llarg del camí. Per tots aquests motius i molts, molts més, t’estic eternament agraïda.

A continuació, m’agradaria donar les gràcies a Ismael Torres. Tu també has estat un bon amic i confident durant aquest viatge. Entre altres coses, m’has mostrat la medicina per a evitar l’estrès i ser una mica més feliç nadant, anant en bici i corrent. Gràcies Isma, per ensenyar-me que amb treball dur i perseverança podem aconseguir totes les metes que ens proposem, i per animar-me a fer coses que mai haguera pensat que jo era capaç de fer.

Una altre incondicional suport i amic ha sigut Bernardo Botella, qui va estudiar l’enginyeria amb mi. A pesar d’haver-te incorporat al nostre grup de recerca en l’etapa final de la meva tesi, m’has ajudat i recolzat tant com t’ha sigut possible. Sempre gaudeixc parlant de tot amb tu. La teva actitud relaxada, la teva paciència, i el teu cor càlid i acollidor han suavitzat els últims clots del camí. Gràcies.

Gràcies Estefanía Serral i Pedro Valderas per les vostres discussions constructives sobre el meu treball, així com l’esforç d’unir-se a mi per escriure treballs de recerca per a promoure el nostre treball. Gràcies a les meves companyes de grup Vicky i Manoli per crear tan bon ambient dins del nostre grup, i als meus companys de laboratori María, Clara, Mario, Salva i Pablo, pels bons moments que hem passat junts, tant al laboratori com fora d'ell. Tots vosaltres teniu un increïble sentit de l'humor; heu mantés les coses lleugeres i a mi amb un somriure. Gràcies Oscar per la teva genial direcció del nostre centre de recerca i Ana per tot el temps que m'has estalviat en el teu treball. També m'agradaria donar les gràcies a tots els membres del centre de recerca ProS, els passats i els presents, per fer-me aquest viatge molt agradable. Gràcies a tots per la vostra amistat, suport i interès en el meu treball.

Tinc la sort de tindre un bon grapat de divertits i bons amics de Benitatxell, València i altres llocs, sense els quals la vida seria més fosca. Aquest treball també ha sigut possible gràcies a ells, que han contribuït d'una manera o un altra a que la meua vida siga més plena i menys estressant, i m'han ajudat a desconnectar de la feina i gaudir del poc temps lliure que aquesta tesi m'ha deixat. Un agraïment especial a Vanesa, per vint i vuit anys de compartir-ho tot amb mi malgrat la distància, i a Marta per ser la meva consciència, la meva fan, camarada de telèfon, etc.

Gràcies a Javi, el meu germà menut, per compartir amb mi la teua visió divertida del món i per mantenir-me sempre somrient. Ocupes un lloc especial en la meva ànima. Tot i que no vivim junts, em preocupe per tu. Gràcies als meus avis, pel vostre afecte per damunt de tot. I un record molt especial a la meua àvia, que encara que no hagi pogut veure el final d'aquesta tesi, estic segura que n'estaria molt orgullosa.

Finalment, els meus pares Joaquin i Teresa; les paraules no poden realment expressar quant vos dec a tots dos. Em donàreu la vida i no heu fet altra cosa que recolzar-me al llarg d'ella. M'heu inculcat moltes qualitats admirables i m'heu donat uns bons fonaments amb els que afrontar la vida. Moltes gràcies pel vostre amor abundant, la vostra confiança en mi, la vostra ajuda i el vostre suport constant. Aquesta tesi és per a vosaltres!

Miriam

Abstract

In Ubiquitous Computing environments, people are surrounded by a lot of embedded services. Since ubiquitous devices, such as mobile phones, have become a key part of our everyday life, they enable users to be always connected to the environment and interact with it. However, unlike traditional desktop interactions where users are used to request for information or input data, ubiquitous interactions have to face with variable user's environment, making demands on one of the most valuable resources of users: human attention. A challenge in the Ubiquitous Computing paradigm is regulating the request for user's attention. That is, service interactions should behave in a considerate manner by taking into account the degree in which each service intrudes the user's mind (i.e., the obtrusiveness degree).

In order to prevent service behavior from becoming overwhelming, this work, based on Model Driven Engineering foundations and the Considerate Computing principles, is devoted to design and develop services that adapt their interactions according to user's attention. The main goal of the present thesis is to introduce considerate adaptation capabilities in ubiquitous services to provide non-disturbing interactions. We achieve this by means of a systematic method that covers from the services' design to their implementation and later adaptation of interaction at runtime.

Models of obtrusiveness and interaction are used to define the interaction obtrusiveness adaptation behavior in a technology-independent way. These models drive the dynamic interaction adaptation by means of an autonomic infrastructure that leverages them at runtime. This infrastructure is capable of detecting changing circumstances (e.g., changes in user location, profile, activity, etc.), and planning and deploying interaction modifications. When a change in the user's situation is detected, services are retargeted to make use of the appropriate interaction components in an automated fashion. Under this autonomic infrastructure, we leverage technology-independent models as if they were the policies that drive the interaction adaptation in a considerate manner.

Furthermore, as user needs and preferences can change over time, we make use of a reinforcement learning strategy in order to adjust the initial obtrusiveness design in a way that maximizes the user's experience. The initial interaction obtrusiveness design ensures a consistent initial behavior according to the user needs at the moment. This design is then adapted for each service to the individual behavior and preferences of users based on user's feedback through experience. Also, models can be further customized by end-users using a mobile interface that allows them to change their own preferences manually.

The proposal has been applied in practice to evaluate it from the designers and the end-users viewpoint. First, the design method has been validated to show its usefulness in helping designers specify this kind of services. Although the development of the services is not completely automated, the guidance offered and the formalization of the involved concepts were proven helpful for designers to develop unobtrusive service interactions. Second, the interaction obtrusiveness adaptation has put in practice with end-users in order to evaluate the user's satisfaction and experience. This validation turned out the relevance of considering obtrusiveness aspects in the interaction adaptation process to enhance user's experience.

Resumen

La Computación Ubicua plantea proveer de inteligencia a nuestros entornos ofreciendo servicios a los usuarios que permitan ayudarlos en su vida cotidiana. Con la inclusión de dispositivos ubicuos en nuestra vida (por ejemplo los dispositivos móviles), los usuarios hemos pasado a estar siempre conectados al entorno, pudiendo interactuar con el. Sin embargo, a diferencia de las interacciones de escritorio tradicionales donde los usuarios eran quienes pedían información o introducían datos, las interacciones ubicuas tienen que lidiar con un entorno de los usuarios variable, demandando uno de los recursos mas valiosos para los usuarios: la atención humana. De esta forma, un reto en el paradigma de computación ubicua es regular las peticiones de atención del usuario. Esto implica que las interacciones de los servicios deberían comportarse de una manera "considerada" teniendo en cuenta el grado en que cada servicio se inmiscuye en la mente del usuario (el nivel de molestia).

Partiendo de las bases de la Ingeniería Dirigida por Modelos (MDE) y de los principios de la Computación Considerada, esta tesis se orienta a diseñar y desarrollar servicios que sean capaces de adaptar sus interacciones de acuerdo a la atención del usuario en cada momento. El principal objetivo de esta tesis es introducir capacidades de adaptación considerada en los servicios ubicuos para proporcionar interacciones que no perturben al usuario. Esto lo conseguimos mediante un proceso de desarrollo que cubre desde el diseño de los servicios hasta su implementación, centrándose en los requisitos de adaptación de la interacción particulares para cada usuario.

Para el diseño del comportamiento de la interacción en base al nivel de molestia se han definido unos modelos de intromisión e interacción independientes de la tecnología. Estos modelos son los que posteriormente conducen la adaptación de la interacción dinámicamente, por medio de una infraestructura autónoma que los usa en tiempo de ejecución. Esta infraestructura es capaz de detectar cambios en la situación del usuario (por ejemplo cambios en su localización, su actividad, etc.) y planear y ejecutar modificaciones en la interacción de los servicios. Cuando se detecta un cambio del contexto del usuario, los servicios se auto-adaptan para usar los componentes de interacción más apropiados de acuerdo a la nueva situación y no molestar al usuario.

Además, como las necesidades y preferencias de los usuarios pueden cambiar con el tiempo, nuestra aproximación utiliza la estrategia del aprendizaje por refuerzo para ajustar los modelos de diseño iniciales de forma que maximicemos la experiencia del usuario. El diseño inicial de la interacción basado en el nivel de molestia nos asegura un comportamiento inicial consistente con las necesidades de los usuarios en ese momento. Luego, este diseño se va refinando de acuerdo al comportamiento y preferencias de cada usuario por medio de su retroalimentación a través de la experiencia de uso. Además, también proporcionamos una interfaz móvil que permite a los usuarios finales personalizarse de forma manual los modelos en base a sus propias preferencias.

El trabajo presentado en esta tesis se ha llevado a la práctica para su evaluación desde el punto de vista de los diseñadores y de los usuarios finales. Por una parte, el método de diseño se ha validado para comprobar que ayuda a los diseñadores a especificar este tipo de servicios. Pese a que el proceso de desarrollo no ofrece una automatización completa, las guías ofrecidas y la formalización de los conceptos implicados ha demostrado ser útil a la hora de desarrollar servicios cuya interacción es no molesta. Por otra parte, la adaptación de la interacción en base al nivel de molestia se ha puesto en práctica con usuarios para evaluar su satisfacción con el sistema y su experiencia de usuario. Esta validación ha desvelado la importancia de considerar los aspectos de molestia en el proceso de adaptación de la interacción para ayudar a mejorar la experiencia de usuario.

Resum

La Computació Ubiqua planteja proveir d'intel·ligència als nostres entorns, oferint serveis als usuaris que permeten ajudar-los en la seva vida quotidiana. Gràcies a la integració de dispositius ubics en la nostra vida (com ara els dispositius mòbils), els usuaris hem passat a estar sempre connectats a l'entorn i a interactuar amb ell. No obstant, a diferència de les interaccions tradicionals d'escriptori on els usuaris eren els qui demanaven informació o introduïen dades, les interaccions ubiqües han de fer front a un entorn dels usuaris variable. Açò esdevé en peticions a un dels recursos més valuosos per al usuaris: la seva atenció. D'aquesta manera, un repte en el paradigma de computació ubiqua és el de regular les peticions de l'atenció de l'usuari. Per tant, les interaccions dels serveis haurien de comportar-se d'una manera "considerada", tenint en compte el grau en que cada servei s'immisceix en la ment de l'usuari (el nivell de molèstia).

Prenent com a base l'Enginyeria Dirigida per Models (MDE) i els principis de la Computació Considerada, aquesta tesi presenta un procés de disseny i desenvolupament de serveis per a que aquests siguen capaços d'adaptar les seues interaccions atenent a l'atenció de l'usuari en cada moment. El principal objectiu d'aquesta tesi és introduir capacitats d'adaptació considerada en els serveis ubics per a proporcionar interaccions que no interrompin a l'usuari. Açò ho aconseguim mitjançant un procés de desenvolupament que cobreix des del disseny dels serveis fins la seua implementació, centrant-se en els requeriments particulars d'adaptació de la interacció per a cada usuari.

Per al disseny del comportament de la interacció en base al nivell de molèstia, s'han definit uns models de intromissió i interacció independents de tecnologia. Aquestos models són els que posteriorment guiaran l'adaptació de la interacció dinàmicament, gràcies a una infraestructura autònoma que els utilitza en temps d'execució. Aquesta infraestructura és capaç de detectar canvis en la situació de l'usuari (per exemple canvis en la seua localització, activitat, etc.) i planejar i executar modificacions en la interacció dels serveis. Quan es detecta un canvi en el context de l'usuari, els serveis s'auto-adapten per a utilitzar els components d'interacció més apropiats d'acord a la nova situació i no molestar a l'usuari.

A més a més, com que les necessitats i preferències dels usuaris poden canviar en el temps, la nostra proposta utilitza una estratègia d'aprenentatge per reforç per a ajustar els models de disseny inicials, de forma que es maximitze l'experiència de l'usuari. El disseny inicial de la interacció en funció del nivell de molèstia ens assegura un comportament inicial consistent en les necessitats dels usuaris amb aqueix moment. Després, aquest disseny va refinant-se d'acord al comportament i preferències de cada usuari mitjançant la seva retroalimentació a través de l'experiència d'ús. A més a més, també proporcionem una interfície mòbil per a que els usuaris puguen personalitzar-se de forma manual els models en base a les seues pròpies preferències.

El treball presentat en aquesta tesi s'ha dut a la pràctica per a la seua avaluació des del punt de vista dels dissenyadors i dels usuaris finals. D'una banda, el mètode de disseny s'ha validat per a comprovar que ajuda als dissenyadors a especificar aquest tipus de serveis. Tot i que el procés de desenvolupament no proporciona una automatització completa, les guies proporcionades i la formalització dels conceptes implicats han mostrat la seua utilitat en el desenvolupament de serveis amb una interacció no molesta. D'altra banda, l'adaptació de la interacció en funció del nivell de molèstia ha estat aplicada en la pràctica amb usuaris finals per a avaluar la seva satisfacció en el sistema i la seva experiència d'ús. Aquesta validació ha revelat la importància de considerar aspectes de molèstia en el procés d'adaptació de la interacció per a ajudar a millorar l'experiència d'ús.

Contents

List of Figures

List of Tables

1

Introduction

Read me first

The reasonable man adapts himself to the world: the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man.

—*George Bernard Shaw (1856-1950). Maxims for Revolutionists.*

The Ubiquitous Computing vision implies a radical paradigm shift in the way users interact with systems (Weiser, 1999). One of its defining traits is the pursuit of *invisibility*, with the purpose of achieving a *calm* world where information seamlessly moves in and out of attention as automation gives way to human interaction (Weiser and Brown, 1997). In this vision, computing resources should become *invisible* to the user in order to allow interaction with the system in a *natural way*. Over the past 20 years, the radical miniaturization and embedding of information, communication, and sensor electronics into almost everything, have made possible to realize this Weiser's vision, providing services that ease people's lives.

Currently, with the trend to connect every physical object ubiquitously to the Internet (also known as the Internet of Things (Gershenfeld et al., 2004)), the possibilities of interaction with the services of the environment have broadened further. For example, customers in a supermarket can be informed about meaningful information such as special offers, friend's opinions about a product or they can even compare prices of products directly interacting with the physical products. However, in an ubiquitous and mobile context where users are permanently connected to the environment, users may be disturbed often (Chittaro, 2010; Schmidt et al., 2012). These problematic interactions are symptoms of the lack of sophistication in their design, in a way that constantly draw the user's attention. This has raised the importance of designing for the "user experience" (Hassenzahl and Tractingsky, 2006; McCarthy and Wright, 2004).

According to the Considerate Computing vision (Gibbs, 2005), user attention is a primary resource to be considered by software systems. This resource is especially relevant in the Ubiquitous Computing paradigm which promotes the natural interaction between the user and the environment. Thus, ubiquitous services should behave in a considerate manner, demanding user attention only when it is actually required according to the user needs and context. As Neil Gershenfeld observed, *there's a very real sense in which the things around us are infringing a new kind of right that has not needed protection until now. We're spending more and more time responding to the demands of machines* (Gershenfeld, 2000).

This work deals with considerate ubiquitous service interactions, which are interactions adaptive to the momentary attentive state of the user. The challenge in an environment full of embedded services is to *provide the right information, at the right time and in the right way for individual users* (Fischer, 2001) avoiding to interrupt them in inopportune situations. If we do not provide support to help control these interactions, users may be disturbed often having a negative impact on their User eXperience (UX)(Law et al., 2009). In this thesis, we tackle this problem by providing a user-centered approach to support self-adaptive unobtrusive interactions in an ubiquitous and mobile context.

The rest of this chapter is organized as follows: Section 1.1 explains

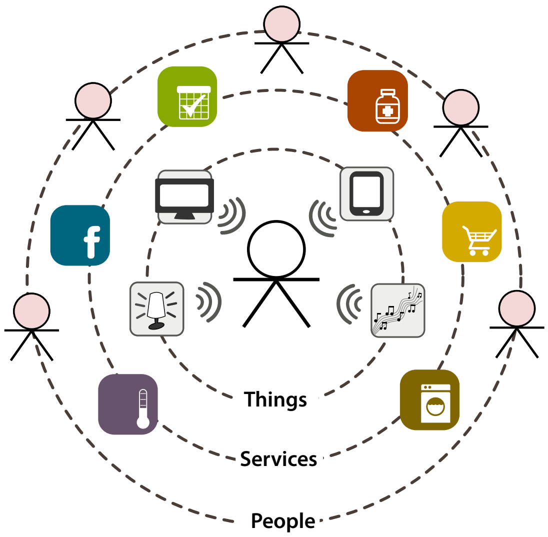


Figure 1.1: Overview of the ubiquitous environment.

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

1.1 Motivation

Emerging ubiquitous 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.), since the use of ubiquitous devices and things (e.g., mobile phones, tablets, TVs, a washing machine, etc.) is widespread nowadays (see Fig.1.1). Unlike desktop software that assumes a full user attention, a static view of the user environment and stable operating conditions where the user initiates the interactions, mobile and ubiquitous environments highlight the importance of letting the user concentrate on his/her tasks by adapting the services to changing user contexts (Hallsteinsen et al., 2012).

Imagine a future in which your fridge announces to you the recipes that can be prepared with the available goods, your TV tells you that your favorite program is beginning and asks you to record it, your mobile agenda is remembering you a meeting to attend; and all of this is happening at the same time. Clearly, living in such an ubiquitous environment on a daily basis may be annoying as users can be frequently interrupted inappropriately (Patterson et al., 2008; Chen and Black, 2008). Conversely, if these services behave in a completely 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 (Tedre, 2006).

With more and more digital services being added to our surroundings, they might be embedded in the actual activities of everyday life resulting in *calm* technology that moves back and forth between the center and the periphery of human attention (Weiser and Brown, 1997). Since user attention is a valuable but limited resource, ubiquitous services must behave in a considerate manner (Gibbs, 2005), requiring user attention only when it is actually necessary. 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 if s/he is attending customers.

The work of Presto (Giner et al., 2010) (a context-aware mobile platform that allows to support different workflows by interacting with the physical environment) highlighted that much of user demands were not related to functional issues, but related with the degree to which interaction intrudes the user's mind (i.e., the obtrusiveness level) instead. Moreover, in many cases, the appropriate obtrusiveness level for interaction with a given service depends on the context and must be changed during runtime. These requirements introduce complexity in the development of mobile and ubiquitous software.

While software engineering is continuously developing methods, tools and best practices, considerate interaction design is still a creative process and yet not suitably supported by engineering practices (Blumendorf et al., 2010a). Existing research addressed issues related to how humans initiate interaction with systems, but we now increasingly observe ubiquitous system designs that also approach humans. Within this reversed interaction, human attention (more than processor speed, communication bandwidth, screen limitations, etc.) becomes the single most critical resource in ubiquitous system design (Ferscha, 2012). Thus, ubiquitous services need to be designed in a way that are capable of being aware of the current user's attentional situation and adapt their interaction with the user according to these attentional resources.

Facing the development of adaptive interactions in terms of obtrusiveness poses the challenge to both express the increasingly complex obtrusiveness and interaction concepts, and to handle the interaction adaptation at runtime. In an environment where the possible combinations of context are constantly increasing, the implementation of ad-hoc solutions to cover all possible combinations is not feasible. In addition, ubiquitous and mobile computing involves a great diversity of technologies and platforms to cover all the possible interactions with the ubiquitous devices that surround users. This heterogeneity forces the developer to know the details of each platform and develop the services according to its capabilities. From a software engineering perspective, there is a need for a user-centered and systematic development method that can free designers and developers from technological details and allow them to regulate the services' interaction obtrusiveness according to the attentional resources of users.

1.2 Problem statement

The adaptation of services' interaction obtrusiveness in an ubiquitous and mobile context has not been fully investigated. The previous discussion indicates that various problems need to be addressed. The work presented in this thesis seeks to improve the design and development of considerate interactions in the Ubiquitous Computing. This challenge is faced from an engineering perspective by considering requirements, design and runtime stages. In particular, the challenges that this thesis addresses can be stated by the following three research questions:

Research question 1. How to capture and manage user's attentional resources in order to model considerate interactions?

Research question 2. How to regulate the interaction obtrusiveness adaptation at runtime in order to provide considerate interactions according to user's situational context?

Research question 3. How to adapt the a priori designed interaction obtrusiveness adaptation to support the changing user preferences over time?

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

1.3 Thesis goals

The main goals of this thesis have been developed to answer the three research questions presented above. Next, we summarize the main contributions of our research work.

First of all, regarding **research question 1**, one of the main goals of this work is the management of user's attentional resources as a theoretical and practical principle for designing considerate interactions in the ubiquitous computing domain. Current user interfaces modeling techniques do not provide support for specifying the dynamic management of the human attention (e.g., not disturbing the user when s/he is engaged in a relevant task). In the present work, user's attention is the focus on designing interactions since it is a limited resource that must be conserved (Gibbs, 2005). In order to manage the obtrusive nature of services' interactions according to user's attention, it is required to define *what* information and capabilities our users require and *how* to provide this information in terms of obtrusiveness. Additionally, from a software engineering perspective, there is a need for the clear definition of the **concepts to be captured and the modeling primitives to capture them**. In order to face this problem, we define a **user-centered design method to capture and manage interaction obtrusiveness requirements** to model considerate interactions.

Regarding **research question 2**, another goal of this work is to regulate the services' interaction obtrusiveness automatically at runtime in order to provide considerate interactions according to the user's situational context. To achieve this, we leverage the design models at runtime to drive the autonomic adaptation of interaction obtrusiveness. That is, we keep the same model representation at runtime that is used at design time. This allows interaction obtrusiveness to be adapted in a technology-independent manner. A **self-regulating autonomic infrastructure** is defined in order to automatically regulate the interaction of the different services to the momentary attentive state of the user following the principles of Autonomic Computing (Kephart and Chess, 2003).

Regarding **research question 3**, one of the goals of the present work is to adapt the a priori design models in order to support the changing user preferences and needs over time. To adjust and improve the initial obtrusiveness design automatically at runtime, we exploit the user's feedback in two ways. First, we follow a reinforcement learning strategy (Sutton and Barto, 1998a) in which our system implicitly learns from user feedback through experience. Specifically, an **obtrusiveness learning system** is defined in order to automatically adapt the interaction obtrusiveness in a way that maximizes the user's satisfaction for a long-term use. This system is built upon the self-regulating infrastructure to adjust its behavior according to the new user preferences. Second, we provide **customization interfaces** to allow end-users to change their own obtrusiveness preferences manually. By means of this capability, we integrate the user in the self-adaptation loop.

The presented contributions form the main building blocks of the **AdaptIO** (Adaptive Interaction Obtrusiveness) framework. This framework is intended to build ubiquitous and mobile systems capable of adapting its services' interaction obtrusiveness according to the user's situational context with the primary goal of enhance the user's experience. We aim to facilitate designers and developers to build this kind of systems by putting the user in the focus of the design.

1.4 Thesis approach

This work is based on Model Driven Engineering (MDE) (Schmidt, 2006) principles in order to design and develop a considerate interaction adaptation of services in a systematic way. MDE proposes the use of models to specify the desired aspects of a system at different levels of abstraction and in a declarative way. A model is a simplification of a system, built with an intended goal in mind, that should be able to answer questions in place of the actual system (Bezivin and Gerbe, 2001). The use of models (such as model of planes in a wind tunnel or models of software systems) in engineering has a twofold benefit. First, models **guide the development** of a system. Second, models allow to **reason about the system** avoiding to deal with technological details.

Sottet in (Sottet et al., 2006) reported the problem of the increasing combinations of users, services, situations and devices to be coped with ad-hod solutions and stressed the relevance of MDE for the modeling of interaction in Ubiquitous Computing systems. Model-based approaches have been developed over the last years introducing well defined development processes to bridge the gap between ubiquitous computing and software engineering. However, interaction design has been often ignored or treated as a secondary concern. Also, we have to include the consideration of user attention aspects and its management as a design principle. In this context, model-driven approaches are growing influence, owing to the promise of yielding more orderly and manageable concepts with enhanced traceability (Constantine, 2009). In this work, we build up on the central ideas of model-driven development (models as first-order citizens) and interaction design (user needs as the focus of the design) to specify and support a considerate interaction adaptation according to user's attention. Specifically, our approach provides the following contributions:

A **user-centered design method** has been defined in order to specify the obtrusiveness degree of services' interactions according to the user's attentional resources and context. In our method, the users' needs drive the design of the services, providing users with personalized services and avoiding overwhelming them. The method has been designed based on the principles of interaction design (Cooper et al., 2007) in order to enhance the user experience.

A **self-regulating autonomic infrastructure** is defined in order to regulate the degree of obtrusiveness required for each service automatically at runtime. This infrastructure exploits the design models at runtime to support the dynamic interaction obtrusiveness adaptation according to the different users' situations.

An **obtrusiveness learning system** is provided in order to readjust the obtrusiveness of services according to user's behavior. It follows a reinforcement learning strategy in order to implicitly adapt the designed interaction obtrusiveness adaptation according to user's feedback, in a way that maximizes the user's satisfaction for a long-term use. Also, customization interfaces are provided to allow users to explicitly change the obtrusiveness design.

1.5 Research methodology

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

The design cycle consists of 5 process steps: (1) awareness of the problem, (2) solution suggestion, (3) development, (4) evaluation, and (5) conclusion. The design cycle is an iterative process; knowledge pro-

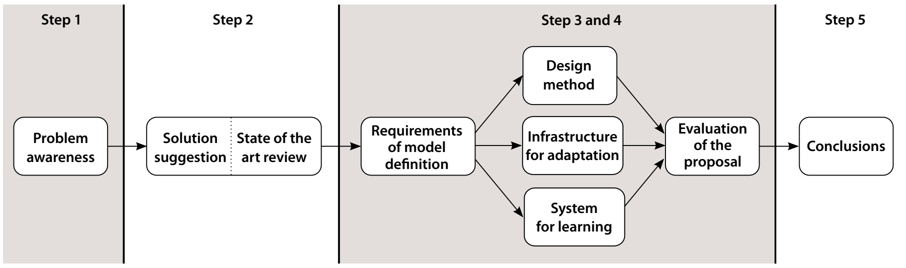


Figure 1.2: Research methodology followed in this thesis.

duced in the process by constructing and evaluating new artifacts is used as input for a better awareness of the problem. Following the cycle defined in the design research methodology, we started with the awareness of the problem (see Fig. 1.2): we identified the problem to be resolved and we stated it clearly.

Next, we performed the second step which is comprised of the suggestion of a solution to the problem, and comparing the improvements that this solution introduces with already existing solutions. To do this, the most relevant approaches from similar application domains were studied in detail. Once the solution to the problem was described, we planned to develop and validate it (steps 3 and 4). These steps are performed in several phases (see Fig. 1.2). The tasks carried out in the step 3 were intended to characterize considerate service's interaction adaptation, define techniques for its design, adaptation at runtime, and learning through experience, providing tool support for each one. When the solution was developed, we evaluated the obtained artifacts of the different phases performed in step 3 and validated the whole approach by applying it to a case study (step 4).

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

1.6 Thesis context

This thesis has been developed in the context of the research center *Centro de Investigación en Métodos de Producción de Software*[1] of the *Universitat Politècnica de València*[2]. The work that has made the development of this thesis possible is in the context of the following research government projects:

- EVERYWARE: Construcción de software adaptativo para la integración de personas, servicios y cosas usando modelos en tiempo de ejecución. CICYT project referenced as TIN2010-18011.
- SESAMO: Construcción de Servicios Software a partir de Modelos. CICYT project referenced as TIN2007-62894.
- “Internet de las Cosas como soporte a Procesos de Negocio”. Primeros proyectos de Investigación de la UPV, referenced as PAID-06-09 number 2920.

1.7 Outline of this thesis

This thesis is presented in nine chapters including this one and two appendices. As a guide to the organization of the remainder of this thesis:

Chapter 2 introduces the main fields that are related to the work that is presented in this thesis in order to provide the reader with a basic background for understanding the overall thesis work.

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

[1]http://www.pros.upv.es
[2]http://www.upv.es

Chapter 4 gives an overview of the thesis work. This overview covers the main building blocks of the approach as well as the process to apply it. In addition, the chapter also introduces how the approach has been evaluated throughout several experiments.

Chapter 5 presents the user-centered design method that is defined to specify the interaction obtrusiveness adaptation of ubiquitous services, and describes the process to be followed at design time.

Chapter 6 describes the mechanisms provided to exploit the design models at runtime and achieve a dynamic interaction obtrusiveness adaptation according to user's situational context. This chapter presents the self-regulating autonomic infrastructure that deals with the dynamic service interaction adaptations.

Chapter 7 presents how the design models can be adjusted at runtime to support the changing user preferences and needs over time by learning it through experience. Also, it introduces the customization tools to allow end-users personalize their obtrusiveness preferences manually.

Chapter 8 details how the proposal has been validated by means of several experiments throughout case studies.

Chapter 9 summarizes the main contributions and publications of this work. In addition, this chapter provides some insights about further work.

Appendix A provides more detail on the specification of the defined modeling language and the tool support to design this kind of systems.

Appendix B shows further details about the case studies used to validate our proposal, and the instruments used in the experiments.

2 Background

The essentials to understand this thesis

Sometimes the problem is to discover what the problem is.
—Gordon Glegg (1969). The Design of Design.

This work deals with the design and development of ubiquitous services that can adapt their interaction at runtime in terms of obtrusiveness according to each user's situation. As it is shown in Fig. 2.1, this work is placed in the intersection of three research areas that have some aspects in common. These disciplines are: *Human-Computer Interaction*, *Context-Aware Computing* and *Considerate Computing*.

This work relies on different concepts and technologies from these areas. In order to clarify the foundations in which our approach relies and provide a basic background for understanding the overall thesis work, different concepts and techniques are introduced in this chapter. Specifically, the rest of this chapter is organized as follows: Section 2.1 provides the main characteristics of the Human-Computer Interaction area. Section 2.2 provides an overview of the Context-Aware Computing area and its principles. Section 2.3 presents the foundations of the

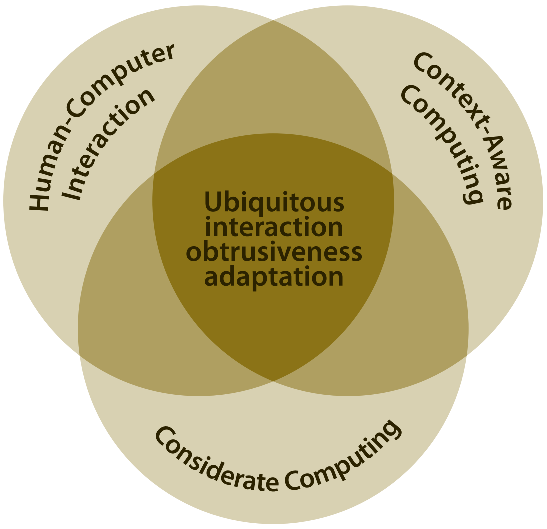

Human-Computer Interaction
Context-Aware Computing
Ubiquitous interaction obtrusiveness adaptation
Considerate Computing

Figure 2.1: Application domains involved in this work.

Considerate Computing paradigm for the development of user interfaces and introduces strategies to minimize interruptions. Finally, Section 2.4 concludes the chapter.

2.1 Human-Computer Interaction

Human-Computer Interaction (HCI) is the study of the interaction between people (users) and computers (Dix et al., 2003). Such interaction is mainly done at the user interface. HCI is inherently multidisciplinary. Among the disciplines involved in HCI, engineering and design methods are relevant. The basic goal of HCI is to improve interactions between users and computers by making computers more usable and receptive to the user's needs.

2.1.1 Design methodologies

A number of diverse methodologies outlining techniques for human–computer interaction design have emerged since the rise of the field in the 1980s. Most design methodologies stem from a model for how users, designers, and technical systems interact. Early methodologies, for example, treated users' cognitive processes as predictable and quantifiable and encouraged design practitioners to look to cognitive science results in areas such as memory and attention when designing user interfaces. Modern models tend to focus on a constant feedback and conversation between users, designers, and engineers and push for technical systems to be wrapped around the types of experiences users want to have, rather than wrapping user experience around a completed system. The most important methodologies are the following:

- **Task-based design:** The main idea behind the method is that a firm understanding of the users' tasks is the proper basis for interactive systems development. With a systematic process of building systems to support users in their tasks, a higher level of usable and useful systems can be achieved by both expert and novice designers. (van Welie, 2001).
- **Activity-centered design:** Activity-centered design is used in HCI to define and study the context in which human interactions with computers take place. Activity-centered design focuses on understanding activities in these contexts. It claims that humans adapt to the tools at hand, and understanding the activities that people perform with a set of tools can more favorably influence the design of those tools (Norman, 2005).
- **Scenario-based design:** This design states that scenarios in HCI help us to understand and to create computer systems and applications as artifacts of human activity. In scenario-based design, descriptions of how people accomplish tasks are a primary working design representation. Software design is fundamentally about envisioning and facilitating new ways of doing things and new things to do (Carroll, 2000).
- **Goal-directed design:** Goal-directed design is an interaction

design methodology created by Cooper (Cooper et al., 2007) that identifies the goals and behaviors of users, and the goals of a business, and directly translates these into design. “Goals are not the same thing as tasks. A goal is an end condition, whereas a task is an intermediate process needed to achieve the goal. . . The goal is a steady thing. The tasks are transient,” he says.

- **User-centered design:** User-centered design (UCD) is a modern, widely practiced design philosophy rooted in the idea that users must take center-stage in the design of any computer system. Users, designers and technical practitioners work together to articulate the wants, needs and limitations of the user and create a system that addresses these elements. Often, user-centered design projects are informed by ethnographic studies of the environments in which users will be interacting with the system (Mao et al., 2001).

This work is based on the user-centered design methodology.

2.1.2 Interaction Modalities

Broadly speaking, there are two extremes of interaction from the intention of the user viewpoint: one in which the user interacts consciously and explicitly with the system; and at the other extreme, the user interacts unconsciously or implicitly. In between, there are various degrees of each kind. Figure 2.2 outlines an interaction model taking implicit and explicit human-computer interaction into account.

Explicit Interaction. In this case, a user interacts with a software application directly by manipulating a GUI, running a command in a command window or issuing a voice command. In short, the user intentionally performs some action. Most interaction with computers, both of the static and mobile variety, is explicit by nature.

Implicit Interaction. Implicit interaction (Schmidt, 2000) is a more recent development and occurs when a user’s subconscious actions

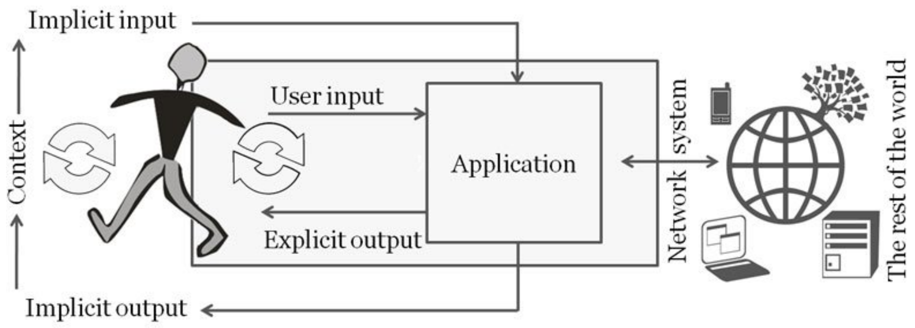


Figure 2.2: Concept of implicit and explicit HCI (Schmidt, 2013)

indicate a preference that may be interpreted as an interaction. It is difficult to capture and subject to error, but it offers useful possibilities in the mobile domain. As a practical example, consider the case of a smart refrigerator. The refrigerator is aware of what products are put inside it. By not returning a product inside could be interpreted by the refrigerator as an indication of a lack of the product. Therefore, the refrigerator could deduce that the user runs out of that product and it could add the product to the shopping list.

Implicit interaction is closely related to user's context (Tamminen et al., 2004) and some knowledge of the prevailing context is almost essential if designers want to incorporate it into their applications (O'Grady et al., 2008). Also, this type of interactions are an inevitable part of what some like to call "smart" products, products whose actions contain some degree of agency, of activity, that occurs without the explicit behest or awareness of the user (Ju, 2008).

Mobile interaction

As the use of mobile devices is widespread nowadays, the design of mobile interactions is taken importance. Chittaro (Chittaro, 2010) stated the aspects that distinguishes mobile interactions from interaction with desktop systems. These aspects difficult to build effective user interfaces for mobile users. Specifically, these aspects are the following.

- **Hardware.** Mobile devices are limited in their input-output capabilities (screen, keyboard, buttons, sound, etc.).
- **Perceptual.** The physical parameters (illumination, noise, temperature and humidity, vibration and motion, etc.) of the mobile user's environment are extremely variable.
- **Motor.** Mobile conditions can impair user's ability to fine control his/her voluntary movements or to take specific postures (e.g., standing).
- **Social.** Social norms related to different environments may make it impossible or unadvisable to use certain modalities (e.g., keeping sound on at a conference is not tolerated).
- **Cognitive.** People in mobility conditions can devote only a very limited attention to interacting with applications on the device.

These issues could be faced by offering a wide repertoire of modalities (and combinations of modalities) to the mobile user, allowing the user to choose those that are socially accepted, easier to perceive and easier to operate in the context s/he is. In this way, multimodality can be a key factor for a better design of mobile interfaces. We take this factor into account to adapt services' interactions and minimize obtrusiveness.

2.1.3 Multimodal interaction

Multimodal interaction provides the user with multiple modes of interfacing with a system. A multimodal interface provides several distinct tools for input and output data.

Sharon Oviatt (Oviatt, 1999) offered a more practical definition, saying that multimodal systems coordinate the processing of combined natural input modalities—such as speech, touch, hand gestures, eye gaze, and head and body movements—with multimedia system output. Matthew Turk and George Robertson (Turk and Robertson, 2000) further refined the difference between multimedia and multimodal systems, saying that multimedia research focuses on the media, while multimodal research focuses on human perceptual channels. They added that multimodal output uses different modalities, such as visual display, audio, and tactile feedback, to engage human perceptual, cognitive, and communication skills in understanding what is being presented. Multimodal interaction systems can use various modalities independently, simultaneously, or by tightly coupling them.

Niels Ole Bernsen (Bernsen, 1994) presented a taxonomy of generic output modalities and their combinations. Zeljko Obrenovic and Dusan Starcevic (Obrenovic et al., 2007) proposed a generic modeling framework for specifying multimodal HCI. All the definitions of multimodal interaction remain on three different concepts:

Modality. It refers to the type of communication channel used to convey or acquire information (e.g., visual, auditory, haptic, etc.). It also covers the way an idea is expressed or perceived, or the manner an action is performed (Nigay and Coutaz, 1993).

Mode. It refers to a state that determines the way information is interpreted to extract or convey meaning (Nigay and Coutaz, 1993).

Multimedia. It focuses on the medium or technology rather than the application or user (e.g., sound clip attached to a presentation). Some examples of media channels are: text, graphics, animation, audio, video, etc. (Buxton, 1990).

Figure 2.3 summarizes the many channels that can be currently exploited to send and receive information from a mobile device. Choosing the appropriate modality or combination of modalities can help in reducing the attentional demand.

In multimodal interactions, multimodal fusion combines the signals of the individual modalities. In addition to combining modalities, the

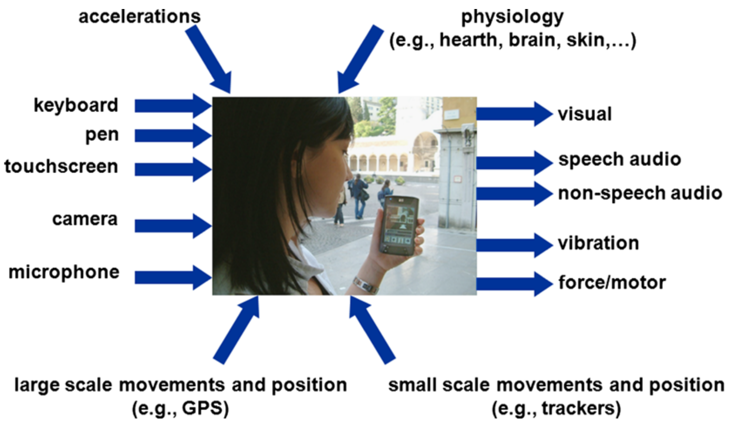


Figure 2.3: Input and output channels in mobile multimodal interfaces (Chittaro, 2010).

fusion extracts semantics and converts the different channels into a common representation. For example, both speech and haptic input may open the route guidance screen in a navigation device by triggering a specific event. Whereas multimodal fusion is beyond the scope of this work, our approach supports multimodal interaction by considering different modalities in the interaction adaptation.

2.1.4 Analysis and discussion

Traditionally, Human-Computer Interaction has been often ignored or treated as a secondary concern when developing software or proposing software engineering methods (Constantine, 2009). Interaction design is still a creative process and yet not suitably supported by engineering practices. Instead of placing users and use at the center of developmental and methodological focus, the dominant modeling languages and methods have relegated them to the periphery.

Furthermore, engineering multimodal mobile interfaces would help designers to address the challenges that mobile interaction presents. With this work, we attempt to fill the gap between software engineering and interaction design by placing users (i.e., human resources) at the focus of the multimodal interaction adaptation design.

2.2 Context-Aware Computing

The idea of *Context-Aware Computing* comes from the vision of Mark Weiser about the Ubiquitous Computing. Weiser envisioned Ubiquitous Computing (UbiComp for short) as a world where computation and communication would be conveniently at hand and distributed throughout our everyday environment (Weiser, 1999). Pervasive Computing (PerCom) (Hansmann et al., 2001), Ambient Intelligence (AmI) (Aarts et al., 2002) or Everyware (Greenfield, 2006) are some of the paradigms that share this goal.

In addition to drawing the fundamental concepts of ubiquitous computing by "vanishing computers into the background", Weiser also shaped the concepts of context-aware computing: "... *computers will come invisible to common awareness. People will simply use them unconsciously to accomplish everyday tasks*". He painted a picture of computers knowing their location, being able to capture information and retrieve context-based information, and offering seamless interaction to support the user's current tasks. Thus, context-aware computing is indeed often related to the research on ubiquitous computing (Dourish, 2004).

In this way, to realize such ubiquitous computing systems with optimal usability, i.e. transparency of use, context-aware behavior is seen as the key enabling factor. Computers already pervade our everyday life - in our phones, fridges, TVs, toasters, alarm clocks, watches, etc - but to fully *disappear*, as in the Weiser's vision of ubiquitous computing, they have to anticipate the user's needs in a particular situation and act proactively to provide appropriate assistance (Schmidt, 2013). This capability requires means to be aware of its surroundings, i.e. context-awareness.

In his 1994 paper at the Workshop on Mobile Computing Systems and Applications (WMCSA), Bill Schilit et al. (Schilit et al., 1994) introduced the concept of context-aware computing and described it as follows:

> "Such context-aware software adapts according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time. A system with these capabilities can examine the computing environment and react to changes to the environment."

The basic idea is that mobile devices can provide different services in different contexts. In addition, they proposed three general categories describing context: *user context*, *physical context*, and *computing context*. Here, the user context describes the situation from the user's viewpoint, including components such as the user's activity and social factors. Physical context includes information of the physical environment, which can be gained with sensor-based measurements. Computing context includes the device connectivity and available computing and application recourses.

The term *context* is widely used with very different meanings. Until now, there is no unequivocal definition of the concept of *context*. There are several definitions of *context* in the literature, but the most used definition in ubicomp was introduced by Dey (Dey and Abowd, 2000):

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

For practical purposes, context is often hierarchically structured describing the relevant features (Schmidt et al., 1999). An example of the feature space is shown in Figure 2.4. By means of this structured space, it becomes easier to link contexts in the real world to adaptations in the system. The advantage of a feature space is that by looking at a set of

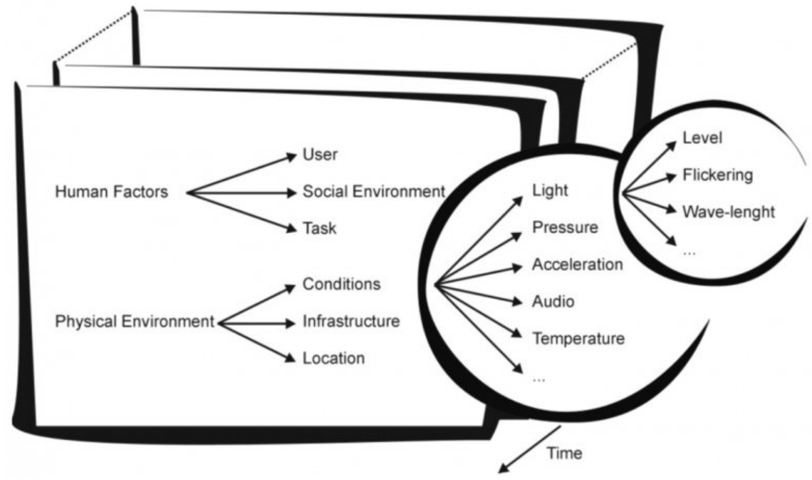


Figure 2.4: Context feature space (Schmidt et al., 1999)

parameters, it can be easily determined if a situation matches a context or not. Knowing which are the factors that should influence the system behavior, one can start to look at how these factors can be determined in the devices. In many cases, this will require sensors that allow the provision of context.

2.2.1 Context of use

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

- *User.* The users of the system who are intended to use the system. In particular, his perceptual, cognitive and action disabilities.

- *Platform.* The computational and interaction device that can be used for interacting with the system. Examples, in terms of resources, include memory size, network bandwidth, screen size, input and output capabilities, and so on.

- *Environment.* The physical environment where the interaction can take place such as noisy environment, lighting conditions, location information, time information, etc.

Under software engineering considerations two kinds can be distinguished (Calvary et al., 2003): *predictive* contexts of use that are foreseen at design time; and *effective* contexts of use that really occur at runtime. Both, the predictive context of use we handle at design time, as well as the effective context of use at runtime are addressed in this work.

2.2.2 Context sensing

A wide variety of sensors are used to acquire contextual information (Mäntyjärvi and Seppänen, 2002; Mostefaoui et al., 2004; Hinckley et al., 2005; Maiden, 2009). Important sensors used are GPS (for location and speed), light and vision (to detect objects and activities), microphones (for information about noise, activities, and talking), accelerometers and gyroscopes (for movement, device orientation, and vibration), magnetic field sensors (as a compass to determine orientation), proximity and touch sensing (to detect explicit and implicit user interaction), sensors for temperature and humidity (to assess the environment), and air pressure/barometric pressure. There are also sensors to detect the physiological context of the user (e.g. galvanic skin response[1], EEG, and ECG).

The challenge to determine context is not only a matter of capture, but also presents a significant inferring challenge. Determining the context with a sufficient confidence summons advanced techniques, and numerous approaches to analyze the data have been proposed. Chen (Chen, 2004) presented three different approaches on how to acquire contextual information. Concerning the way data is captured, sensors can be classified in three groups (Indulska and Sutton, 2003).

Physical sensors. These are the most frequently used type of sensors.

[1]Galvanic skin response measures the resistance between two electrodes on the skin. The value measured is dependent on how dry the skin is.

Many hardware sensors are available nowadays which are capable of capturing almost any physical data. Some examples of physical sensors have been described above.

Virtual sensors. Virtual sensors source context data from software applications or services. For example, it is possible to determine an employee's location not only by using tracking systems (physical sensors) but also by a virtual sensor, e.g., by browsing an electronic calendar, a travel-booking system, emails, etc., for location information.

Logical sensors. These sensors make use of a couple of information sources, and combine physical and virtual sensors with additional information from databases or various other sources in order to solve higher tasks. For example, a logical sensor can be constructed to detect an employee's current position by analyzing logins at desktop PCs and a database mapping of devices to location information

In this work, we make use of physical and virtual sensors.

2.2.3 Context models

In order to define and store context data in a machine processable form, a *context model* is needed. There are several context modeling approaches (Bettini et al., 2010) such as *key-value models*, *markup scheme models*, *graphical models*, *object oriented models*, *logic based models*, and *ontology based models*. However the evaluation presented in (Strang and Linnhoff-Popien, 2004) shows that ontologies are the most expressive models and fulfill most of the requirements.

Thus, in order to capture and reason about context, we use ontologies. In a widely-quoted definition, an ontology is "a specification of a conceptualization" (Gruber, 1993). An ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. Thus, it allows a programmer to specify in an open and meaningful way the concepts and relationships that collectively characterize some domain.

An ontology mainly contains the following elements:

- *Classes*: which are all kinds of existences or concepts. A class usually refers to a collection or a category of objects sharing some common character and well accepted under common sense.
- *Attributes*: properties that identify a class itself from other classes.
- *Relationships*: a relation between two ontology classes interprets how the two classes, more precisely the objects of these classes, are related. Typically a relation is a particular connection between two classes that specifies how an object is connected to the other in an ontology.
- *Individuals*: instances or objects of the defined classes. All the objects under a category are named as "individuals" of this class.

Numerous tools are available to define declarative representations and to publish and share ontologies developed by the World Wide Web Consortium, e.g., the Resource Description Language (RDF)[2] and the Web Ontology Language (OWL)[3].

2.2.4 Context-Awareness as enabler for Autonomic Computing

Context-awareness its a property that enables the vision of *Autonomic Computing*. In 2003, Kephart and Chess (Kephart and Chess, 2003) warned that the dream of interconnectivity of computing systems and devices could become the "*nightmare of pervasive computing*" in which architects are unable to anticipate, design and maintain the complexity of interactions, leaving such issues to be dealt with at runtime. They stated that the only option is *Autonomic Computing*.

Specifically, *Autonomic Computing* refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity to operators and users (Horn,

[2]http://www.w3.org/RDF
[3]http://www.w3.org/TR/owl-features/

2001). Started by IBM in 2001, this initiative ultimately aims to develop computer systems capable of self-management, to overcome the rapidly growing complexity of computing systems management, and to reduce the barrier that complexity poses to further growth.

In a self-managing autonomic system, the human operator takes on a new role: instead of controlling the system directly, s/he defines general policies and rules that guide the self-management process. For this process, IBM defined the following four functional properties (Kephart and Chess, 2003):

- **Self-configuration.** Automatic configuration of components.
- **Self-healing.** Automatic discovery, and correction of faults.
- **Self-optimization**. Automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements.
- **Self-protection.** Proactive identification and protection from arbitrary attacks.

To achieve autonomic computing, IBM suggested a reference model for autonomic control loops which is sometimes called the MAPE-K (Monitor, Analyse, Plan, Execute, Knowledge) loop and is depicted in Figure 2.5. This loop is composed by an *autonomic manager* software element and a (hardware or software) *managed element*. The autonomic manager continually executes a *monitor-analyze-plan-execute* loop in which it observes sensor readings, analyzes and plans an appropriate management decision, and then executes that decision via effectors. A central *knowledge base*, accessible by the other components, contains knowledge pertaining to the likely effectiveness of various possible management decisions in achieving the manager's overall policy objectives. Typically, such knowledge takes the form of an explicit system model.

Even though the purpose and thus the behavior of autonomic systems vary from system to system, every autonomic system should be able to exhibit a minimum set of properties to achieve its purpose:

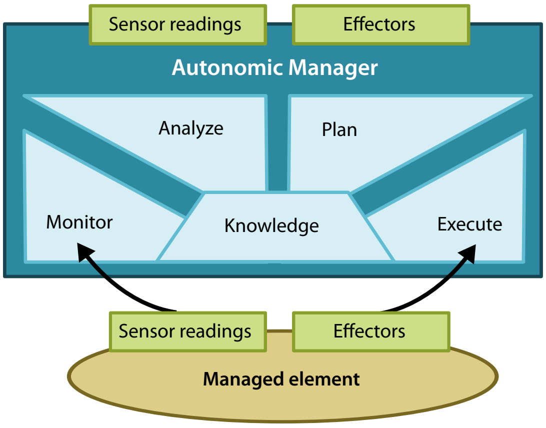


Figure 2.5: IBM's MAPE reference model for autonomic control loops

Automatic. This essentially means being able to self-control its internal functions and operations. As such, an autonomic system must be self-contained and able to start-up and operate without any manual intervention or external help. Again, the knowledge required to bootstrap the system (know-how) must be inherent to the system.

Adaptive. An autonomic system must be able to change its operation (i.e., its configuration, state and functions). This will allow the system to cope with temporal and spatial changes in its operational context either long term (environment customization/optimization) or short term (exceptional conditions such as malicious attacks, faults, etc.).

Aware. An autonomic system must be able to monitor (sense) its operational context as well as its internal state in order to be able to assess if its current operation serves its purpose. Awareness will control adaptation of its operational behavior in response to context or state changes.

One important aspect to be considered in these systems is the difficulty of engineering a sufficiently accurate knowledge module that can achieve acceptable performance in deployed systems (Tesauro, 2007). Tesauro highlighted this problem, especially for model-based approaches, since today's computing systems are continually evolving and periodic redesign of the knowledge modules are necessary. To overcome this knowledge bottleneck he proposed to apply *learning* techniques.

2.2.5 Reinforcement Learning

Pattie Maes argued that to gain trust in context-aware systems, the end user must be involved in the specification of the system's behavior, but s/he usually cannot directly program it (Maes, 1994). In addition, user's habits evolve through time (Byun and Cheverst, 2001a), implying the need of repeated modifications of the context-aware knowledge. Learning techniques has been proposed as a possible solution for those problems. In particular, reinforcement learning has been considered as a solution for getting "in context" qualitative feedback from the user.

Reinforcement Learning (RL) (Sutton and Barto, 1998a) is an approach where an agent acts in an environment and learns from its previous experiences to maximize the sum of rewards received from its action selection. The reward is classically a continuous function between -1 and 1. 1 corresponds to satisfaction, -1 disapproval and 0 means no opinion.

RL is used to establish policies obtained from observing management actions. At its most basic, it learns policies by trying actions in various system states and reviewing the consequences of each action (Sutton and Barto, 1998a). In RL's normal operation, illustrated in Figure 2.6, an agent learns effective decision-making policies through an online trial-and-error process in which it interacts with an environment. Assuming time progresses in discrete steps, each interaction consists of:

- observing the environment's current state s_t at time t,
- performing some legal action a_t in state s_t, and
- receiving a reward r_t (a numerical value that the user would like to maximize) followed by an observed transition to a new state s_{t+1}.

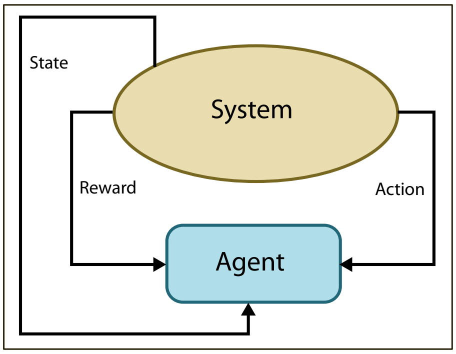


Figure 2.6: A standard reinforcement learning interaction loop

The advantage of reinforcement learning is that it does not require an explicit model of the system being managed, hence its use in autonomic computing (Littman et al., 2004; Dowling et al., 2006). However it suffers from poor scalability in trying to represent large state spaces, which also impacts on its time to train. To this end, a number of hybrid models have been proposed which either speed up training or introduce domain knowledge to reduce the state space (Tesauro, 2007; Whiteson and Stone, 2006).

2.2.6 Analysis and discussion

Summarizing the results from context-aware computing, the goal of using context information is to make interaction with a system more relevant, useful and robust. Once the system has recognized in which context an interaction takes place, this information can be used to change, trigger, and adapt the behavior of applications and systems.

However, usually the attentional resources of the user have not been considered to design context-aware systems. This result in interruptions at inopportune moments. In this work, we use the context in a different way in order to minimize interruptions to the user. We consider the attentional demand (e.g., human limitations) as a first class concept in order to adapt the services.

Furthermore, we integrate the autonomic computing principles for building context-aware services. Autonomously regulating the obtrusiveness of context-aware services based on context information and obtrusiveness models provides high potential for offering more non-disturbing services to the user.

2.3 Considerate Computing

"Humanity has connected itself through roughly three billion networked telephones, computers, traffic lights -even refrigerators and picture frames- because these things make life more comfortable and keep us available to those we care about. So although we could simply turn off the phones, close the e-mail program, and shut the office door when it is time for a meeting, we usually do not. We just endure the consequences" (Gibbs, 2005).

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

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

Designers and engineers should design computing devices that negotiate rather than impose the volume and timing of their communications with the user. Cooper and Reimann's About Face 3 describe some of the most important characteristics of considerate interactive products:

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

For example, a mobile phone that behaves like a considerate person knows that, after you have completed a call with a number that is not in your contacts, you may want to save the number, and provides an easy and obvious way to do so. An inconsiderate phone forces you to scribble the number on the back of your hand as you go into your contacts to create a new entry.

Since user attention is a valuable but limited resource, an environment full of embedded services must behave in a considerate manner, demanding user attention only when it is actually required. The increase of information overflow and continuous request for user attention (Horvitz et al., 2003) make interruptions a common occurrence in ubiquitous environments. A few researchers have attempted to define interruptions and establish a taxonomy that describes the different issues surrounding interruptions (Ginsburg, 2004). McFarlane (Mcfarlane, 1997; McFarlane, 1999; McFarlane and Latorella, 2002) and Latorella (Latorella, 1998) defined the concept of human-computer interruption as follows:

Interruption. An interruption can be defined as an unanticipated request for task switching from a person, an object, or an event while multitasking. Interruptions typically request immediate attention and insist on action, and reduce productive focus.

Disruption. If an interruption is allowed to distract the user into action, it escalates into disruption (Jackson et al., 2001). Thus, disruption is defined as the negative effects on a primary task from interruptions requiring transition and reallocation of attention focus from one activity to another.

Distraction. Distraction affects people's memories. On returning to a suspended activity, it is possible for them to have forgotten where they were in the activity. As a result, they may not restart from where they left off but will recommence at a different point of entry (Preece et al., 1994).

McFarlane (Mcfarlane, 1997; McFarlane, 1999; McFarlane and Latorella, 2002) developed a taxonomy of human interruption, as a tool for answering interruptions research questions. The taxonomy lists eight dimensions of human interruption. Manipulating each dimension can influence the disruptive effects of interruptions. These dimensions and examples of the dimensions are the following:

- *Source of interruption:* self; another person; computer.
- *Individual characteristic of person receiving interruption:* limitations of: perceptual processors, cognitive processors, motor processors, memory, focus of consciousness, processing streams; willingness and ability to be interrupted.
- *Method of coordination:* immediate; negotiated; mediated; scheduled.
- *Meaning of Interruption:* alert; stop; divert attention (task-switching); distribute attention (task-sharing); remind; communicate information.
- *Method of expression:* physical (i.e., verbal); type (i.e., by purpose).
- *Channel of conveyance:* face-to-face; mediated by a person; mediated by a machine.
- *Human activity changed by interruption:* conscious or subconscious; individual activities; joint activities.

- *Effect of interruption:* change in activity; change in memory; change in awareness; change in focus of attention; loss of control over activity.

2.3.1 Receiving and managing interruptions

Generally, interruptions in the ubiquitous computing take the form of notifications that are received on the various devices that a user may possess or perceive in his/her environment (Ramchurn et al., 2004). To this end, the four main ways identified by McFarlane to disrupt someone are (mentioned above as methods of coordination):

- **Immediate:** the person must leave current task to attend to interruption. It requires the attention of the user immediately without any other choice.
- **Negotiated:** the interruption is announced to the person, and then the person decides when to attend to it. It allows the user to choose the moment when they will deal with the interrupting activity that needs attention.
- **Mediated:** the interruption is announced to the person's personal digital assistant (or another third party), which determines when the best time is to interrupt the person. It alerts the user on another device rather than the one on which it was supposed to be delivered.
- **Scheduled:** the person is interrupted during prearranged times only.

Whatever the form in which a message is received, there are four possible responses to it (Clark, 1996):

1. *Take-up with full compliance.* Handle the interruption immediately.
2. *Take-up with alteration.* Acknowledge the interruption and agree to handle it later.

3. *Decline.* Explicitly refuse to handle the interruption.
4. *Withdraw.* Implicitly refuse to handle the interruption by ignoring it.

In each of the above responses, some degree of mental processing by the user is involved in deciding what course of action to take. In most cases the answer depends on the preferences of the user with respect to the information available about the content of the notification.

2.3.2 Contextual factors that influence interruption

According to Ho and Intille (Ho and Intille, 2005), there are at least 11 factors that impact the perceived burden of an interruption. These factors are:

- *Activity of the user.* The activity the user was engaged in during the interruption.
- *Utility of the message.* The importance of the message to the user.
- *Emotional state of the user.* The mindset of the user, the time of disruption, and the relationship the user has with the interrupting interface or device.
- *Modality of interruption.* The medium of delivery, or choice of interface.
- *Frequency of interruption.* The rate at which interruptions are occurring.
- *Task efficiency rate.* The time it takes to comprehend the interruption task and the expected length of the task.
- *Authority level.* The perceived control a user has over the interface or device.
- *Previous and future activities.* The tasks the user was previously involved in and might engage in during the future.

- *Social engagement of the user.* The user's role in the current activity
- *Social expectation of group behavior.* Activities and expected reaction to interruption of nearby people.
- *History and likelihood of response.* The type of pattern the user follows when an interruption occurs.

Fischer et al. (Fischer et al., 2010) also introduced the distinction between *local* and *relational* contextual factors.

Local contextual factors. These factors describe the individual's context at the moment of the interruption. They have been distinguished into two categories: *cognitive* and *social* factors (Grandhi and Jones, 2009). *Cognitive factors* that have been studied include the person's mental workload when interrupted, or the person's current activity. *Social factors* are those that discern the impact of the social surrounding on the interruptibility of the interrupted person, e.g., the presence of other people, or the impact of specific organizational or cultural norms of the place where the person is interrupted on how interruptions are managed.

Relational contextual factors. These factors take into account properties of the interruption itself. Fischer et al. (Fischer et al., 2010) suggest that relational factors should be taken into account in addition to local contextual factors when trying to estimate a person's receptivity. The consideration of *who* (sender/source), the *what* (content), and the characteristics of the channel the interruption is delivered in (*in what channel*) may impact receptivity or the management of the interruption. Lastly, *what effect* the interruption may have could also be an important factor in predicting receptivity.

2.3.3 Cognitive aspects of interruptions

Numerous studies have investigated the disruptive effects of interruptions. It is well known that interruptions affect behavior. These effects are related to limitations in a person's cognitive abilities to work during interruptions. In the following subsections, we discuss the cognitive aspects of visual attention and task switching as the effects of interruptions.

Visual attention

Attention can be regarded as a cognitive selection process, depending heavily on the user goals and tasks (Wood et al., 2006). Attention can either be voluntarily directed towards or involuntary grabbed by objects, sounds or movement in the environment. The challenge is how to make sure the user is focusing the limited attentional resources on the right task. In order to make accurate estimates on user attention, we need a system that can sense, model and adapt to the attentive states of the user.

Research in this direction has been addressed under the term of Attentive Interfaces; interfaces that can prioritize information presentation based on reasoning about information processing resources (i.e. attention) (Vertegaal, 2003; Jameson, 2002). These interfaces model momentary attentive states of the user based on sensing techniques. Adaptations in system behavior are guided by sensing gaze direction, body posture, hand posture and speech (Vertegaal, 2002; Zhai, 2003).

Task switching

Usually, interruptions can cause task switching. Interruptions can be caused by the environment (distraction), by the system (notification) or by the user (changing attentional focus). After finishing the other or secondary task, the user returns to the original task. According to Oulasvirta and Saariluoma (Oulasvirta and Saariluoma, 2004), in the case of task-switching, information must be saved into long-term working memory before the switch occurs, if the interrupted task is to be resumed at a later time.

Task switching influences attention in two ways. First, the interruption of a task causes attention to be distracted away from that task. Depending on user goals, this is either an unwanted distraction from the primary task or an attraction to valued and necessary information (McCrickard and Chewar, 2003). Unwanted interruption has negative consequences for task performance, causing disorientation on part of the user (Nagata, 2003). Second, research indicates that the process itself of switching between tasks places high demand on attention, thereby increasing cognitive workload. In addition, a switch between tasks is more disruptive than sequential finishing of tasks and thereby affects performance (Neerincx et al., 2000).

2.3.4 Analysis and discussion

Existing researches have been presented with the same goal: make the software behave like a considerate human being. Commonly, interactive products irritate us because they are not considerate, not because they lack features.

However, the existing considerate systems are primarily focused on issues of perception and interpretation of attention, not too much concern is spent on the nature of the resultant actions. In addition, existing approaches provide conceptual frameworks to obtain design guides, but they lack of tools for the development of this kind of systems and methodological guidance.

The conclusion is that these initiatives are almost exclusively focused on evaluating the adequate timing for interruptions, while interaction adaptation that addressed in this work has received few attention.

2.4 Conclusions

The purpose of this chapter was to provide a brief introduction to the existing background on top of which this work is built on. The analysis has considered three application domains: *Human-Computer Interaction*, *Context-Aware Computing* and *Considerate Computing*. This work is aimed at providing development support for systems that fit in these three areas. Thus, much of the technologies and techniques introduced are applied in the following chapters. A detailed analysis of the proposals that are more close to the goal of this work is provided in Chapter 3.

3 State of the Art
Understanding the need

All intelligent thoughts have already been though; what is necessary is only try to think them again.

—*Johann Wolfgang von Goethe (1749-1832).*

This chapter introduces the most important approaches that support the design, development and runtime support of service's interaction obtrusiveness adaptation. Once we have analyzed in Chapter 2 the general application domains in which this work fits, we analyze the specific proposals in these domains that are closely related to our approach. This analysis allows us to determine the way in which each proposal addresses the aspects that are central in our approach.

The ubiquitous interaction obtrusiveness adaptation can be considered in the intersection of *Human-Computer Interaction*, *Context-Aware Computing* and *Considerate Computing* domains. Figure 3.1 illustrates these different research areas that are relevant to the present work and their subdomains situated in the intersection of the main areas. In particular, the approaches considered deal with systems that fulfill most of

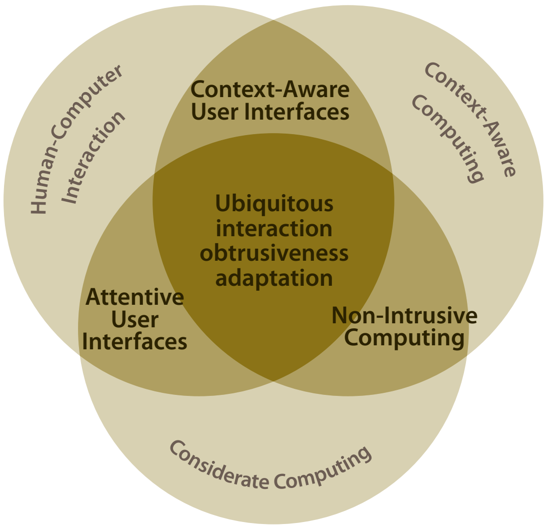

Human-Computer Interaction
Context-Aware User Interfaces
Context-Aware Computing
Ubiquitous interaction obtrusiveness adaptation
Attentive User Interfaces
Non-Intrusive Computing
Considerate Computing

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

the features that characterize our approach. We identified three research areas where interaction obtrusiveness adaptation fits in some aspects: *Context-Aware User Interfaces*, *Attentive User Interfaces*, and *Non-Intrusive Computing*. Relevant approaches in these areas have been analyzed and discussed in this chapter.

Next, we first introduce a set of important dimensions to characterize and classify the different approaches. Then, Section 3.2, Section 3.3, and Section 3.4 describe the approaches related with our work that are placed on each one of the introduced subdomains. For each one of the subdomains, we provide an analysis and discussion of it at the end of each section. After this analysis, in Section 3.5, we present a discussion about the existing literature and explain the most important benefits of our approach. Finally, Section 3.6 concludes the chapter.

3.1 Analysis criteria

In order to classify the different approaches and determine to what extent they can support the *interaction obtrusiveness adaptation* of services, we have identified the basic issues to consider, according to which we will study the current approaches. In this section, we define the criterion and potential dimensions to be fulfilled when developing this kind of systems. These dimensions have been defined from previous criteria in the literature. However, there is no specific criterion for interaction obtrusiveness adaptation. Therefore, we identified the relevant dimensions in previous criteria and adjusted a set of dimensions according to the focus of this research.

According to the existing research in the design and development of adaptive user interfaces, there are a number of related work that present criteria to be applied to various user interface adaptation aspects (Höök, 2000; Dey and Häkkilä, 2008; Paramythis et al., 2010; Paternò et al., 2012). For example, Höök defined a design criterion for intelligent user interfaces (Höök, 2000), Dey and Häkkilä defined several design guidelines to build context-aware applications (Dey and Häkkilä, 2008), Paramythis et al. defined a layered evaluation of adaptive systems and presented an evaluation criterion in terms of adaptation phases (Paramythis et al., 2010), and Paterno et al. (Paternò et al., 2012) defined a UI adaptation-oriented criterion. In particular, we present the criteria in a three groups following the division of (Paternò et al., 2012). It is worth pointing out that different grouping criteria can be also used. However, the selected criteria is specifically focused on user interface adaptation, which is close related to the requirements to achieve the interaction obtrusiveness adaptation of services. Due to our work is based on the foundations of model-driven engineering, we have also included dimensions of this paradigm. The UI adaptation-oriented criteria with their dimensions are the following:

Technical criteria. Criteria that can be objectively measured by just considering the support offered by the system. According to this criteria, the following dimensions have been identified:

- **Flexibility in adaptation.** Adaptation approaches may be *adaptive*, *adaptable*, or based on a *mixed-initiative*. This dimension determines the degree of adaptation of the system from adaptability (allow users to customize their system from a predefined set of parameters) to adaptivity (automatically perform adaptations without a deliberate user request), while mixed-initiative approaches are a mix of both.

- **Multimodality support**. This dimension refers to the possibility to support a ”graceful combination of several modalities”.
- **Integration capabilities.** It describes if the system supports a consistent view of users, services, and devices to provide a seamless integration of all the components and allows all entities to interact with each other in an easy way.
- **Use of models at runtime.** This dimension simply indicates whether the approach uses models at runtime or not.
- **Customizability of the adaptive behavior.** The aim of this dimension is to understand the extent of user control over the adaptive process, namely the user's ability to control both the circumstances that lead to triggering adaptation, and how adaptation is actually performed and applied. It ranges from only controlled by the system (the system decides what is going to be adapted) to explicitly controlled by the end-users (the end-users keep the control about what have to be changed and in which conditions), or by both.
- **Support for open/extensible adaptation at runtime.** This dimension evaluates whether it is possible to introduce new adaptation rules at runtime, or the system is not open to support dynamically the addition of new adaptive behavior.
- **Required training.** This dimension describes whether or not the system requires a period of training before it can be run. It has been identified to avoid the cold-start problem (Serral, 2011).

User-oriented criteria. Criteria that refer to the involvement of the user on a number of aspects connected with adaptation. The relevant dimensions identified for this criteria are the following:

- **User preferences support.** This dimension indicates whether or not the approach takes into account the user preferences in the specification of the system behavior, and therefore, in the adaptation process.
- **Obtrusiveness of the adaptation.** With this dimension we want to understand to what extent the adaptation exploits an appropriate level of obtrusiveness depending on the current context. Thus, this dimension indicates whether or not the approach takes into account obtrusiveness aspects to define the interaction adaptation.
- **User participation in the specification.** This dimension refers to the degree of user participation in the system specification. It indicates if the approach has an active involvement of users in the system specification.
- **Personalization for individual needs.** This dimension indicates whether the system supports a dynamic personalization based on user preferences after the system is deployed.

Modeling-related criteria. Criteria related to the adaptation which is specifically related to the activity of modeling. The extracted dimensions for this criteria are the following:

- **Tool support.** With this dimension we want to check if tools are provided for specifying the system at design time (e.g., adaptation rules, interaction components, etc.), for performing adaptation at runtime, and for customizing adaptation rules at runtime.
- **Well-defined methodology support.** This dimension indicates if there is a clear definition of the activities involved to specify the system and their coordination in order to integrate all the parts into a unified design.
- **Adaptive behavior modeling.** This dimension aims to check the technique that the approach uses for specifying the adaptive behavior.

- **Adaptation rules definition.** This dimension simply indicates the way in which the approach specifies the adaptation rules.

The presented dimensions are summed up for each related work using a table. In order to better study the differences between each work and the goals of this thesis, we have grouped the identified key dimensions according to the confronted research questions into three blocks. Thus, for each one of the different approaches, we present the following information:

Dimensions related with the *user-centered design method*:

- User preferences support
- Adaptive behavior modeling
- Obtrusiveness of the adaptation
- Tool support
- Well-defined methodology support
- User participation in the specification

Dimensions related with the *runtime support for interaction obtrusiveness adaptation*:

- Flexibility in adaptation
- Multimodality support
- Integration capabilities
- Adaptation rules definition
- Use of models at runtime
- Required training

Dimensions related with the *runtime obtrusiveness learning*:

- Customizability of the adaptive behavior
- Personalization for individual needs

- Support for open/extensible adaptation at runtime

As well as the above explained dimensions, this table also includes the application domain property to characterize the domains in which the approaches are applied and check whether they are independent of the domain. Next, we describe the most important approaches related with our work paying special attention on the key dimensions identified above.

3.2 Context-Aware User Interfaces

Context-Aware User Interfaces (Schmidt, 2013) use context to make interactions more relevant and useful for creating a user experience that is tailored specifically to each context. They may include both output and input as well as various modalities.

Different modeling languages have been proposed in this area to formalize distinct aspects of the user interface and the interaction. Calvary et al. in (Calvary et al., 2003) give an overview of different modeling approaches to deal with user interfaces supporting multiple targets such as PDA's, work stations, etc. Other approaches such as UsiXML (Limbourg et al., 2004) or TERESA (Mori et al., 2004) define domain-specific languages that are designed from the beginning to deal with the description of user interfaces in a device-independent manner. An extension of UML, named CUP 2.0 (den Bergh and Coninx, 2006) was introduced for high-level modeling of context-sensitive interactive applications. All these approaches use the models to generate multiple variants of static, final user interface code, which is then executed. Conversely, our research focus is the dynamic interaction adaptation at runtime. Relevant proposals in this direction are introduced below.

Cameleon-RT. Cameleon-RT (Balme et al., 2004) is a conceptual architecture reference model constructed to define the problem space of user interfaces released in ubiquitous computing environments. Their reference model covers user interface distribution, migra-

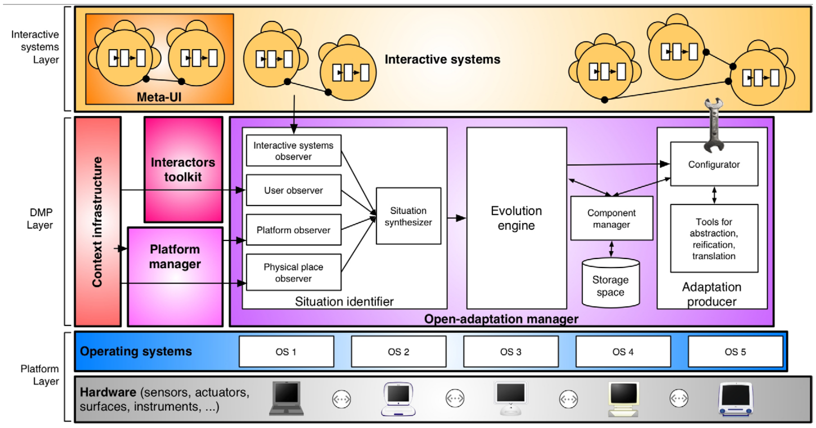


Figure 3.2: The Cameleon RT architecture reference model (Balme et al., 2004).

tion, and plasticity (adaptable to context and still usable).

As shown in Figure 3.2, the architecture defines three layers of abstraction. The platform layer includes the physical hardware and the legacy operating systems. The DMP-middleware layer provides a context infrastructure for context management, a platform manager and the interaction toolkit to support resource discovery, and an open-adaptation manager that supports the UI adaptation. The interactive systems (including a Meta-User Interface) are contained in the top layer. This Meta-User Interface provides metadata about the user interfaces of all of the DMP-middleware components and allows users to control the state of them.

Framework for Adaptive Multimodal Environments (FAME). FAME (Duarte and Carriço, 2006) is a model-based Framework for Adaptive Multimodal Environments. The framework's objective is to guide the development of adaptive multimodal applications. The architecture proposed by FAME (see Fig. 3.3) uses a set of models to describe relevant attributes and behaviors regarding user, platform and environment. The information stored in these

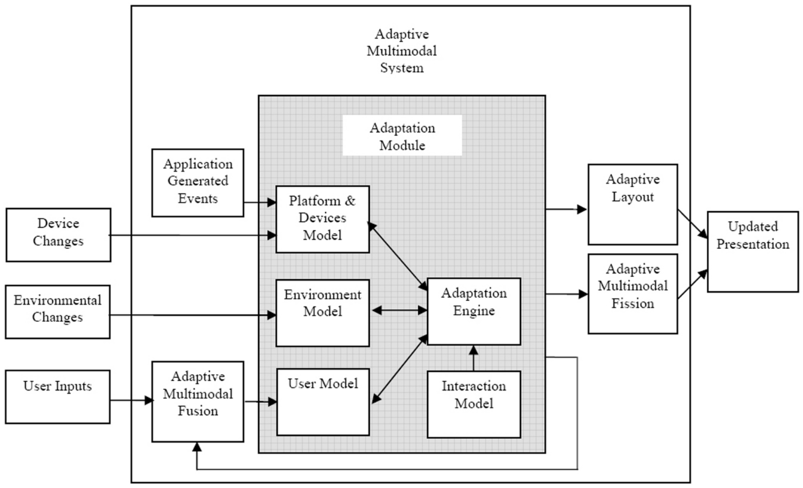


Figure 3.3: Architecture of FAME (Duarte and Carriço, 2006).

models, combined with user inputs and application state changes, is used to adapt the input and output capabilities of the interface. It introduces a behavioral matrix to assist in the adaptation rules development. The matrix reflects the behavioral dimensions in which a user can interact with an adaptable component. A set of guidelines systematizes the development process.

Two levels are identified in the architecture of FAME. The inner level (Adaptation Module) is responsible for updating the models and generating the system actions. The outer level (Adaptive Multimodal System) is responsible for the multimodal fusion of user inputs, the transmission of the application specific generated events to the adaptation core, the execution of the multimodal fission of the system actions, and the determination of the presentation's layout.

The adaptation is based on three different classes of inputs: user actions (from the input devices), application generated events and device changes, and environmental changes (acquired by sensors). The adaptation rules are defined in a behavioral matrix for each

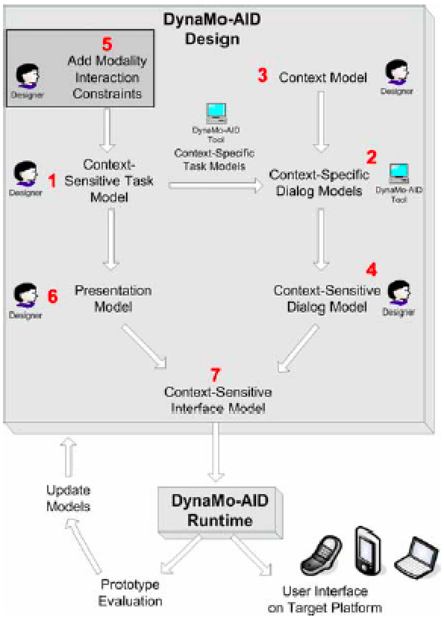


Figure 3.4: DynaMo-AID Development Process (Clerckx et al., 2008).

adaptable component. There are three values in each cell of the behavioral matrix: the first element defines the current rules in use; the second element stores the number of times the rule has been applied; and the third element defines a threshold for rule activation.

DynaMo-AID. DynaMo-AID (Dynamic Model-Based User Interface Development) (Clerckx et al., 2008) is a design process and a runtime architecture to develop context-aware user interfaces that support dynamic context changes. DynaMo-AID is part of the Dygimes (Coninx et al., 2003) User Interface Creation Framework.

Figure 3.4 shows an overview of the DynaMo-AID development process. The process begins with the specification of models, then, a supporting tool generates a prototype taking into account these models, and after evaluating the prototype, the models can be updated until the designer is satisfied with the user interface. Finally, the user interface can be deployed on the target platform.

The design process is composed by: a *context-sensitive task model*; *dialog models* that are extracted from the context-sensitive task model; a *context model* denoting what kind of context information can influence a context change; a *context-sensitive dialog model* that connects the context model and the dialog models; a *presentation model* that describes how the interaction should be presented to the user; and the *context-sensitive interface model* that is the union of the previously model components. Also, the task model is annotated with *Modality Interaction Constraints* to support the appropriate modalities for each task at runtime.

XUL-based User Interface Framework. Butter et al. (Butter et al., 2007) developed a XUL-based user interface framework to adapt mobile user interfaces to the different screen resolutions and input capabilities of the heterogeneous mobile devices. This framework is an extension to XUL that allows the easy creation of applications which adapt themselves to different devices and user context. XUL is implemented as a XML dialect; it allows graphical user interfaces to be written in a similar manner to web pages. It relies on multiple existing Web standards and Web technologies, including CSS, JavaScript, and DOM.

This framework is based on a XUL renderer for different Java configurations/platforms and control the user interface via plain Java classes. To match the current context with the appropriate appearance (specified using Cascading Style Sheets), XUL is extended to allow loading a CSS depending on a pre-specified user context. Then, the selection of the relevant stylesheets is done dynamically so the layout and appearance of the UI changes automatically if the user context changes.

The Multi-Access Service Platform (MASP). MASP (Blumendorf et al., 2010b) is a model-based runtime system for the creation of Ubiquitous User Interfaces. It is based on a set of models to adapt the user interfaces dynamically to the current context of

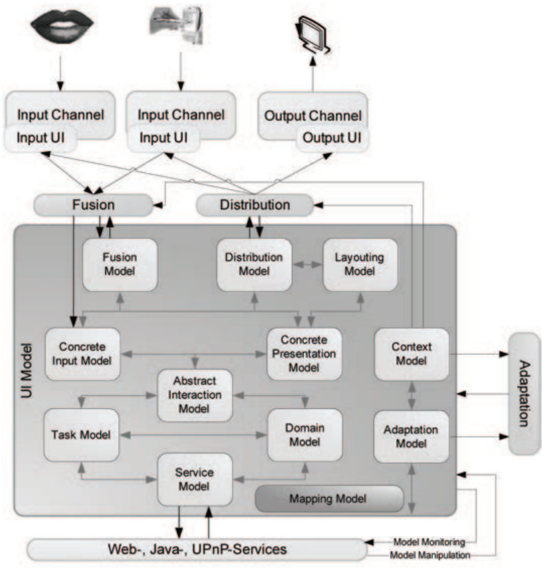


Figure 3.5: The MASP Runtime Architecture (Blumendorf et al., 2010b).

the user. MASP separates four levels of abstraction: *Task and Domain Model*, *Abstract Interaction Model*, *Concrete Interaction Model*, and the *final user interface* derived from the modeled information.

Figure 3.5 shows an overview of the MASP architecture comprehending the set of models, their relationships as well as the components to provide advanced functionality. While the *Service Model* allows interaction with backend systems, the *Task and Domain Model* describe the basic concepts of the underlying user interface. Based on these concepts, the interaction model (*Abstract*, *Concrete Input*, and *Concrete Presentation Model*) define the actual communication with the user. A *Context Model* provides interaction-relevant context information, while a *Fusion and Distribution Models* allow the definition of fusion and distribution rules to support multimodal and distributed interfaces. Finally, a *Layouting Model* allows the definition of layout constraints.

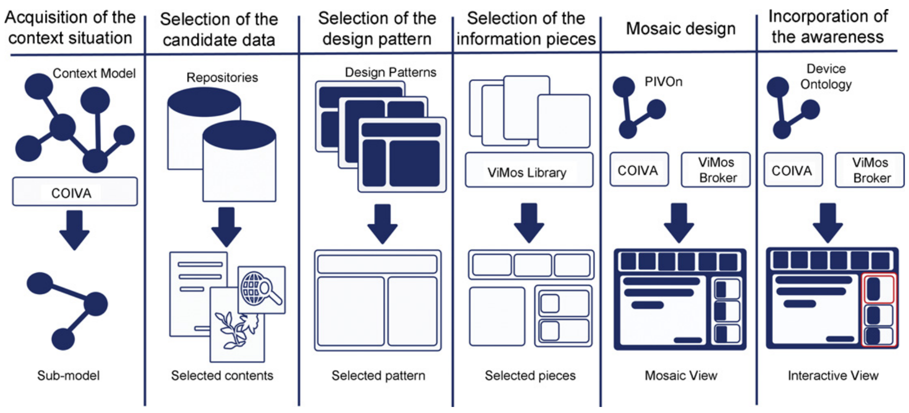


Figure 3.6: Generation process of the ViMos framework (Hervás and Bravo, 2011).

MASP is based on a client-server architecture. The resources are connected via channels and each user interface delivered to a resource is adapted according to the capabilities of the resource. Multimodality is supported by a distribution component and a fusion component.

Visualization Mosaics (ViMos). The ViMos framework (Hervás and Bravo, 2011) is an infrastructure to generate context-powered information visualization services dynamically. ViMoS is an information visualization service that applies context-awareness to provide adapted information to the user through embedded devices in the environment. The displayed views are called mosaics, that are developed as user interface widgets.

Figure 3.6 shows the mosaic generation process to generate user interfaces at runtime. This process uses the COIVA architecture (Hervas, 2009) which provides the information needed to generate the user interface dynamically. COIVA includes a reasoning engine that hastens the start-up process, enabling the automatic generation of ontological individuals. The process that follows the ViMos framework is the following: first, it acquires the context situation by means of a sub-model of valid individuals; then, the significant items of content to be offered to the user are selected for a specific visualization service; after that, the design patterns for that visualization service are also selected; next, the ViMos broker selects the container widgets (information pieces) that are appropriate for visualizing the candidate data and the mosaic is designed; finally, ViMos recognizes general events that cause changes in an information piece or in the general view.

3.2.1 Analysis and discussion

The different approaches in the context-aware user interfaces field are mainly focused on adapting the user interfaces to the technical limitations of the device (e.g., screen resolution). These proposals normally define the adaptation space in terms of the environment (e.g., location), the user (e.g., user disabilities), and the platform (e.g., screen size). Conversely, our approach introduces the notion of obtrusiveness to manage the attentional resources of the user by defining the adaptation space in terms of obtrusiveness. We address a different issue that is more related to human limitations (e.g., user attention).

Table 3.1 shows a comparison of the presented approaches. The table shows to which extent the different approaches cope with the identified key dimensions to characterize the interaction obtrusiveness adaptation of services. All the introduced approaches have the capacity of adapting the user interfaces dynamically at runtime (see *flexibility in adaptation* row).

Regarding the *user-centered design method*, some of them (DynaMo and XUL) do not take into account the user preferences in the specification of the adaptive behavior. Each one of the approaches specifies the adaptive behavior of the system (i.e., variability) in a different way (e.g., by using conceptual graphs, interaction models, ontologies, etc.). However, all of the solutions 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

Property	**CAMELEON**	**FAME**	**DynaMo**	**XUL**	**MASP**	**ViMos**
User pref. support	yes	yes	no	no	yes	yes
Adap. behav. modeling	conceptual graphs	interaction model	interface model	CSS	interaction models	visualization ontology
Obtrusiv. of adaptation	no	no	no	no	no	attention level
Tool support	no	no	yes	no	yes	yes
Well-defined methodology	no	yes	yes	no	no	no
User participation	no	no	no	no	no	no
Flexibility in adaptation	adaptive	adaptive	adaptive	adaptive	adaptive	adaptive
Multimod. support	no support	high	no support	no support	high	no support
Integration capabilities	systems & devices	users & devices	services & devices	devices	services & devices	users, services & devices
Adaptation rules	graph of situations	behavioral matrix	annotated constraints	context tags	adaptation model	behavior rules
Runtime models	no	no	no	no	yes	no
Training	no	no	no	no	no	no
Customizab.	no	system	no	no	no	no
Personaliz. indiv. needs	no	yes	no	no	no	no
Open adaptation	no	no	for devices	no	no	no

Table 3.1: Related work from the context-aware user interfaces perspective

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.

Although some of the techniques have tool support to help designers in the specification of the system, only two of them (FAME and DynaMo) provide a well-defined methodology that guide designers in the system definition at each step of the process. Also, none of them are based on a user-centered design in which the user participates in the design process by evaluating the system before development and giving feedback to improve it. Conversely, our approach provides a well-defined methodology supported by the appropriate tools letting users participate in the system design by means of a simulation phase. In this phase, the users evaluate the design and give feedback to designers in order to improve the initial design.

With regard to the *runtime interaction obtrusiveness adaptation*, only two of the presented approaches (FAME and MASP) support a multimodal interaction adaptation. Also, only ViMos supports a consistent view of users, services and devices by means of an ontological description of them (see *Integration capabilities* row). Our approach provides an integration of users, services and devices by means of the autonomic infrastructure that allows all entities to interact with each other in an easy way. In order to specify the adaptation rules that trigger the interaction adaptation, the proposals use various mechanisms such as a graph of situations, a behavioral matrix, annotated constraints in the interface model, context tags in the CSS, an adaptation model composed by context situations and adaptations, and behavior rules. Conversely, our approach defines the adaptation rules by means of transitions in the unobtrusive adaptation space. In this way, designers can define how the interaction style should be adapted according to user's attentional resources by making use of declarative specifications.

In order to adapt the user interfaces at runtime, most of the approaches mainly aim at the utilization of the static design models to support the generation of user interfaces at runtime. The connection between the models and the system is implicitly hidden within the implementation and they do not interpret the models explicitly. Contrary to the others, MASP interprets the models at runtime to address the dynamics of user interface adaptation. Our approach also leverages the design models at runtime to support the dynamic interaction obtrusiveness adaptation. However, our approach also modifies these models at runtime in order to improve the initial design to achieve a better user experience. In this way, the interaction adaptation can be re-adjusted over time while the system is being executed and it is re-adapted while running. As the different approaches define the adaptation rules a-priori, they do not need training to adapt the system (see *training* row).

Since the presented approaches are mainly concerned with adaptation aspects, they lack of learning capacity to improve the initial adaptation design over time (see *learning* rows). Only FAME supports the learning of user preferences by storing the past user interactions. However the learning technique is not specified.

3.3 Attentive User Interfaces

Computing interfaces that are sensitive to the user's attention are called *Attentive User Interfaces* (AUIs) (Vertegaal, 2003). Attentive User Interfaces can also be used to display information in a way that increase the effectiveness of the interaction by generating only the relevant information (Huberman and Wu, 2007). Roel Vertegaal of Queen's University has been pioneering the design of user interfaces that are attentive to their users. These attentive user interfaces aims to optimize the allocation of the attentive resources of users and systems. Cues of user attention are applied to make devices more sociable and efficient and to increase the bandwidth of user communication with and via computers. Relevant research in this area is detailed below.

Active Messenger. The Active Messenger (AM) project (Schmandt et al., 2000) is a server-based agent that manages the user's incoming e-mail messages, prioritizes them, and forwards them to the available communication channels (e.g., pagers, fax machines, phones, etc.) in order to minimize interruption in the deliver. The forwarding rules are specified in a user preference file and can be modified by the agent to adjust to the user's current situation. First, the agent sends the message to the first channel and checks the status of the message and the channel. If the user has not read the message after a certain time, the agent sends it to the next appropriate channel that is available, and so forth.

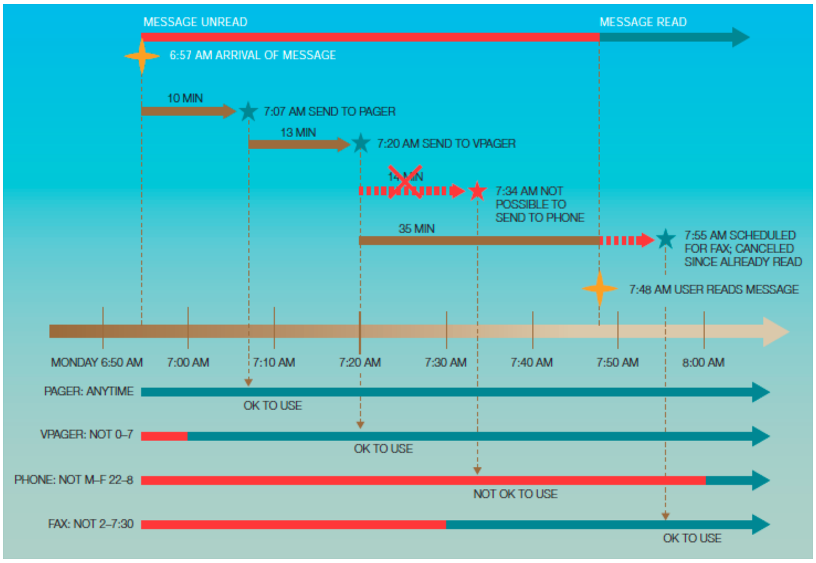


Figure 3.7: Channel seleccion in the Active Messenger process (Schmandt et al., 2000).

The agent process is based on a single Perl script. Each channel is added as a new module with a configuration file that the user edits during setup to express his/her preferences. This file could be updated via a Web-based interface. In this file, the user can also define channel sequence: rules that map each class of message to a set of devices or delivery mechanisms. Figure 3.7 shows an example of the channel selection process.

Models of attention in computing and communication. Horvitz et al. (Horvitz et al., 2003) proposed the use of *Bayesian models* to sense and reason about the user's attention in order to identify ideal decisions in messaging, interaction, and computation. They constructed by hand and learned from data Bayesian models, working to reveal current or future attention *under uncertainty* from an ongoing stream of clues. Bayesian attentional models take as inputs sensors that provide streams of evidence

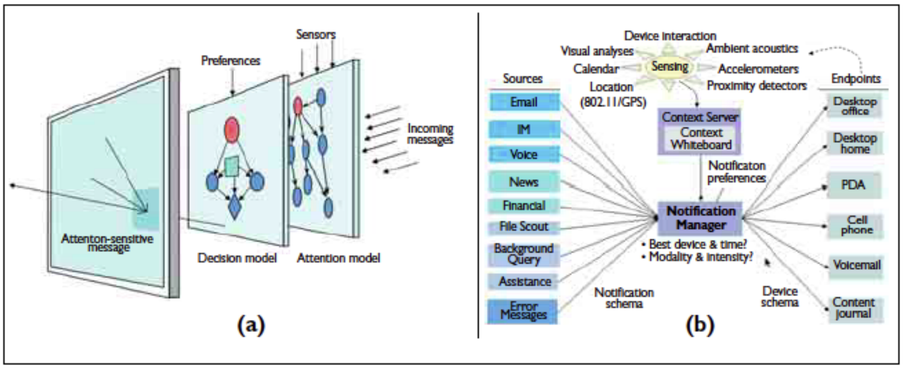


Figure 3.8: Overview of the Notification Platform (Horvitz et al., 2003).

about attention and provide a means for computing probability distributions over a user's attention and intentions.

Also, they presented the *Notification Platform* (NP) in order that computers and communication systems have awareness of the value and costs of relaying messages, alerts, and calls to users. The *Notification Platform* system modulates the flow of messages from multiple sources to devices by performing ongoing decision analyses (see Fig. 3.8). These analyses balance the expected value of information with the attention-sensitive costs of disruption. The system employs a probabilistic model of attention and executes ongoing decision analyses about ideal alerting, fidelity, and routing.

A Framework for designing attentive notification systems. McCrickard and Chewar (McCrickard and Chewar, 2003) presented a theoretical framework for designing *attentive notification systems* (ANS) by means of modeling user notification goals. They stated that the success of a notification system hinges on accurately supporting attention allocation between tasks, while simultaneously enabling utility through access to additional information. They recognize four general sources of utility that can

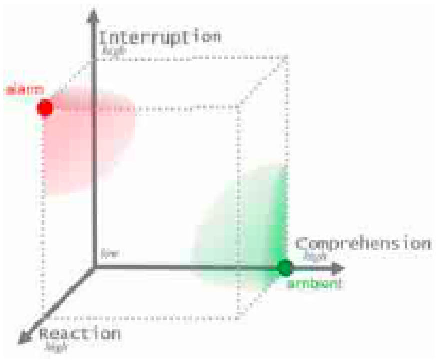


Figure 3.9: Overview of the framework reflecting the user goals (McCrickard and Chewar, 2003).

result from associated user goals: comprehension, reaction, interruption, and satisfaction. The first three general goals can be viewed as critical parameters that can be benchmarked to reveal design process.

Figure 3.9 shows the framework that depicts the three critical parameters. The axis scales correspond to the level of importance for user places on benefits resulting from each parameter. This framework provides the foundation for a conceptual model of user notification goals that can improve design decisions for notification systems.

A Framework for Attentive User Interfaces. Vertegaal et al. (Vertegaal et al., 2006) presented a framework for augmenting user attention through attentive user interfaces. They extended the GUI elements to interactions with ubiquitous remote devices, drawing parallels with the role of attention in human turn taking. This is shown in Fig 3.10. Windows and icons are supplanted by graceful increases and decreases of information resolution between devices in the foreground and background of user attention; devices sense whether they are in the focus of user attention by observing presence and eye contact; menus and alerts are replaced by

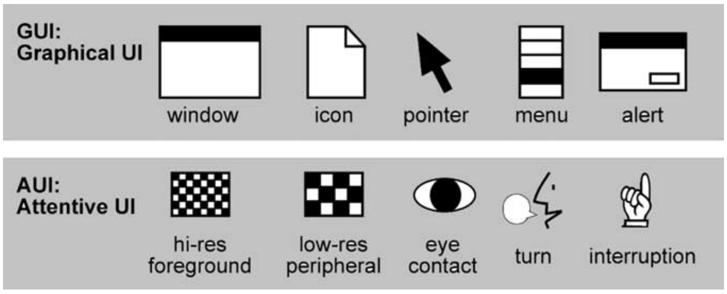


Figure 3.10: Equivalents of GUI elements in Attentive UI (Vertegaal et al., 2006).

a negotiated turn taking process between users and devices. Such characteristics and behaviors define an attentive user interface.

The framework is based on five key properties of AUIs: (1) *Sensing attention* by tracking user's physical proximity, body orientation and eye fixations; (2) *Reasoning about attention* by statistically modeling simple interactive behavior of users; (3) *Communication of attention* to other people and devices; (4) *Gradual negotiation of turns* to determine the availability of the user for interruption; and (5) *Augmentation of focus* with the goal of augmenting the attention of their users.

Personal attentive user interfaces. Streefkerk et al. (Streefkerk et al., 2006) proposed the design of personal attentive user interfaces (PAUI) for which the content and style of information presentation were based on models of relevant cognitive, task, context and user aspects. They presented a user-centered design (UCD) method for the iterative development and validation of the proposed PAUI. The relevant cognitive aspects considered were attention, working memory, task switching, and situation awareness. Regarding to task aspects, they analyzed information access, prioritization, and notification. Context aspects taken into account were location, time and environment. Finally, the relevant user aspects were preferences, duties, expertise and individual differences.

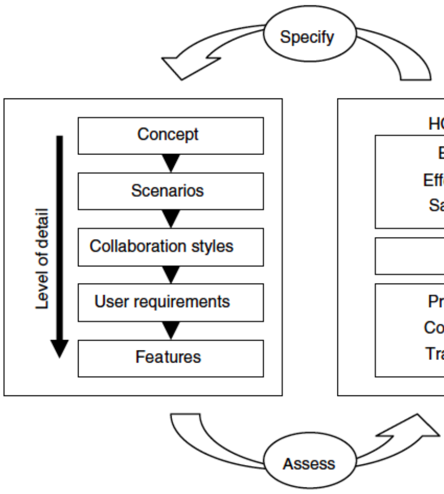


Figure 3.11: The user-centered design process (Streefkerk et al., 2006).

The approach was based on the usability engineering approach which incorporates scenario-based design (Carroll, 2000). The process (shown in Fig 3.11) starts with the definition of a concept, which is a general description of the proposed system. Then, a scenario is drafted from the relevant usage context. From this scenario, collaboration styles are defined, which indicate how the system interacts with the user. Next, user requirements are derived. These requirements will be based on the relevant cognitive, task, and context aspects. Finally, collaboration styles and user requirements form the basis of the features. Assessment of the collaboration styles, user requirements and features is done by validating them to objective quality criteria, such as established HCI metrics.

AuraOrb. AuraOrb (Altosaar et al., 2006) is an ambient notification appliance that deploys progressive turn taking techniques to minimize notification disruption (see Fig 3.12). It uses information about user interest to determine its notification strategy. To detect user interest, AuraOrb uses eye contact sensing in an initially

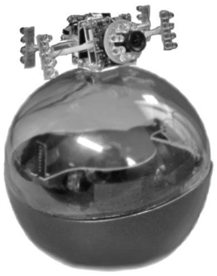


Figure 3.12: AuraOrb (Altosaar et al., 2006).

ambient light display. When eye contact is detected, AuraOrb displays the subject heading of the notification, i.e., in the center of the user's attention. Touching the orb causes the associated message to be displayed on the user's last attended computer screen. When user interest is lost, AuraOrb automatically reverts back to its idle state.

The design principles that guided the development of AuraOrb were: 1) avoid foreground display, 2) sense user interest across devices, and 3) negotiate user attention.

3.3.1 Analysis and discussion

The different approaches in the attentive user interfaces area are mainly focused on detecting or inferring user attention for enhancing computing and communication systems. One of the main problems that may arise is that the sensing technologies used for sensing attention may be invasive. Also, as mentioned in (Vertegaal et al., 2006), the most pressing issue related to these sensing technologies of attentive user interfaces is privacy. It is difficult to safeguard privacy of the user when devices routinely sense, store and relay information about their identity, location, activities, and communications with other people.

Table 3.2 shows a comparison of the presented approaches. All these

Property	**AM**	**NP**	**ANS**	**AUI**	**PAUI**	**AuraOrb**
User pref. support	yes	yes	yes	yes	yes	yes
Adap. behav. modeling	no	bayesian inference model	no	no	scenarios	no
Obtrusiv. of adaptation	yes	yes	yes	yes	yes	yes
Tool support	no	no	no	no	no	no
Well-defined methodology	no	no	no	no	yes	no
User participation	yes	no	no	no	yes	no
Flexibility in adaptation	adaptive	adaptive	adaptive	-	adaptive	adaptive
Multimod. support	no support	no support	no support	no support	no support	no support
Integration capabilities	users & devices	users, services & devices	users & presentation modes	users & devices	users & features	users
Adaptation rules	configuration files	decision model	attention-utility	priority models	collaboration styles	behavior rules
Runtime models	no	no	no	no	no	no
Training	no	no	no	no	no	no
Customizab.	no	no	no	no	no	no
Personaliz. indiv. needs	manually	manually	no	no	no	no
Open adaptation	no	no	no	no	no	no
Domain	e-mail messages	incoming messages	ambient & alarm	-	police office	workstation

Table 3.2: Related work from the attentive user interfaces perspective

approaches define systems and frameworks to manage user's attention (see *obtrusiveness of adaptation* row) and adapt interactions to the current attentive state of the user according to user preferences (see *user* *pref. support* row). Variability modeling is partially supported in two approaches (NP and PAUI) by means of creating probabilistic attentional models and scenarios based on context (see *adap. behav. modeling* row). However, none of the approaches provide neither modeling languages nor tools for the development of this kind of attentive user interfaces. They lack of methodological guidance (only PAUI provides a design process) and do not provide tool support for the easy system specification. Also, only PAUI follows a user-centered design process where the user participates in the design specification. Conversely, our approach provides a well-defined methodology and tools for the design and development of interaction obtrusiveness adaptation of services in a declarative manner by following a user-centered design process.

While these approaches explore how to mitigate the problem of interruptions by adapting the interaction to the different communication channels (see *flexibility in adaptation* row), they work over a limited set of message sources and delivery channels. Also, they are limited to one communication channel at a time (multimodality is not supported). Conversely, our work aims to give the capability of interaction adaptation to the different pervasive and mobile services in a centralized way to preserve user attention, and provides a multimodal interaction delivery by means of different interaction modalities.

A common factor in all of these approaches is that they support a consistent view of the status of users (see *integration capabilities* row) to provide a personal interaction. Some of them support different devices or presentation modes, and can integrate different services. Moreover, in order to define the adaptation rules, the different approaches use different kind of mechanisms (from configuration files to behavior rules, see *adaptation rules* row), and none of them require a training period before deployment since the behavior is defined at design time. However, the adaptation provided by these proposals is not based on models at runtime (see *runtime models* row). Generally, they have a system component that analyses the user's context and took the decisions based on the adaptation rules. Our approach leverages the design models at runtime as if they were the policies that drive the interaction obtrusiveness adaptations. In this way, the design effort made at design time is not only useful for producing the system but also provides a richer semantic base for self-adaptation during runtime.

Regarding the learning over time, some of the proposals (AM and NP) allow users to update his/her own preferences manually by means of a user interface (see *personaliz. indiv. needs* row). However, this is not the research focus of these approaches. Finally, it is worth noticing that most of the approaches are tailored for specific domains, such as workstations or e-mail messages and cannot interoperate with each other.

3.4 Non-Intrusive Ubiquitous Computing

In a strongly connected ubiquitous computing environment, users face a large number of interactions. Some of them are desired while others are not, usually depending on how busy the users are and what those interactions are about. These are contributing to feelings of information overload and unexpected interruptions in undesired situations, such as delivering a private picture to the whiteboard during a meeting. Thus, the focus of the *Non-Intrusive Computing* is helping users to control these interactions and devices in order to avoid their frustration. The approaches studied from this area are detailed below.

Toward More Sensitive Mobile Phones. Hinckley and Horvitz (Hinckley and Horvitz, 2001) described sensing and interaction techniques intended to help designers make mobile phones more polite and less distracting. They modeled interruptibility by considering the user's likelihood of response and the previous and current activity.

Three sensors necessary to detect these factors were built in a mobile computing device: a two-axis linear accelerometer for tilt sensing, a capacitive touch sensor to detect if the user was holding the device, and an infrared proximity sensor that detected if the head was in close proximity to the device. The experiments performed include choosing an appropriate notification modality

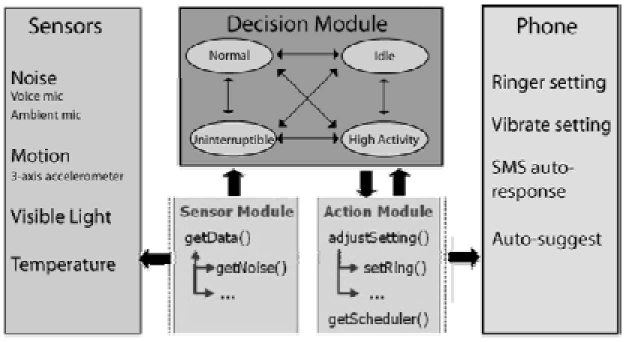


Figure 3.13: Arquitecture of SenSay (Siewiorek et al., 2003).

for an incoming call and reducing attentional demands to answer the calls.

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

To provide context information SenSay uses light, motion, and microphone sensors. The sensors are placed on various parts of the human body with a central hub, called the sensor box, mounted on the waist. SenSay introduces the following four states: *uninterruptible*, *idle*, *active*, and the default state, *normal.* A number of phone actions are associated with each state. For example, in the uninterruptible state, the ringer is turned off. SenSay can either eliminate unwanted interruptions or actively notify the user of an incoming call by adjusting ringer and vibrate settings. It also has the ability to relay the user's contextual information to the caller when the user is unavailable. The components of SenSay are shown in Fig. 3.13.

Minimising Intrusiveness in Pervasive Computing Environ-

ments. Ramchurn et al. (Ramchurn et al., 2004) proposed an agent-based negotiation system for managing intrusiveness in a meeting environment. It mainly concerns how to display a message when there is a meeting. This system defers the handling of messages to software agents that represent their owners.

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

Bayesphone. Bayesphone (Horvitz et al., 2005) is a call handling system that predicts whether users will attend meetings on their calendar and the cost of being interrupted by incoming calls. It is composed by two applications: 1) a desktop application that performs inference, cost-benefit analyses, and value of information precomputation of ideal real-time actions and inquiries, and 2) a mobile application that downloads the precomputed policy file from the desktop via a device synchronization program.

The desktop application analyzes each meeting, making inferences with a Bayesian network models for both attendance and interruptability (see an example in Fig. 3.14). The client application considers these inferences along with the costs of deferral of calls from callers in different groups, the expected cost of interruption with taking of calls for each forthcoming meeting, and the history of incoming calls in the user's call log. Finally, it precomputes the ideal call-handling actions and interactions for each meeting.

Reducing the Perceived Burden of Interruptions. In this study (Ho and Intille, 2005), a context-aware mobile computing device was

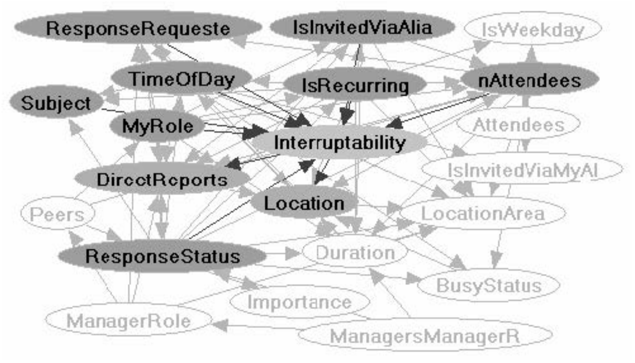


Figure 3.14: Bayesian network learned (Horvitz et al., 2005).

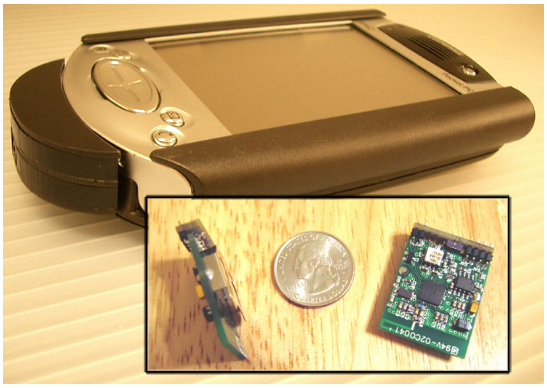


Figure 3.15: The 3-axis wireless accelerometers (Ho and Intille, 2005).

developed that automatically detects postural and ambulatory activity transitions in real-time using wireless accelerometers (see Fig. 3.15). This device was used to experimentally measure the receptivity of interruptions delivered at activity transitions relative to those delivered at random times.

The hypothesis was the possibility that prompts from mobile devices may be perceived as less disruptive if they are presented at times when the user is transitioning between different physical

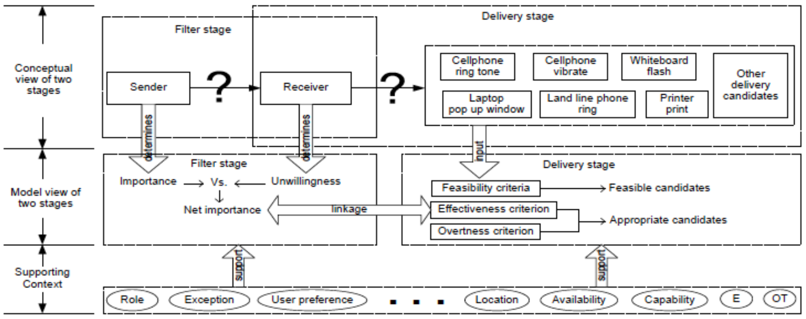


Figure 3.16: Overall view of the non-intrusive computing (Chen and Black, 2008).

activities. The experiment was motivated by the casual observation that a transition between two different physical activities may strongly correlate with a task switch, and a task switch may be a better time to prompt the user with an interruption than an otherwise random time.

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

A Quantitative Approach to Non-Intrusive Computing. Chen and Black (Chen and Black, 2008) proposed an approach to make ubiquitous computing non-intrusive so that users can focus on their tasks. They addressed the problem of intrusion of interactions directly by means of models of importance and willingness to better choose the appropriate communication mechanism to deliver the message.

Specifically, the proposed model involves four essential factors in two stages: importance and willingness in the filter stage and effectiveness and overtness in the delivery stage. They quantify these factors and link the two stages through the net importance. Figure 3.16 summarizes the modeling of the non-intrusive computing approach.

Context-Aware Mobile Phone Profiles. Valtonen et al. (Valtonen et al., 2009) presented a context-aware, proactive and adaptive phone-profile control system that automatically adapts the profile to the best alternative base on the current context. The system is based on fuzzy control and allows an automatic control of the active phone profile based on place, time, and weekday.

The control starts with fuzzification in which each membership function receives a degree of membership. The system can then use all the input variables' fuzzy sets in fuzzy reasoning, giving a phone profile as a fuzzy set. The system constantly analyzes the user's context and proactively adjusts the active phone profile according to the learned rules when it recognizes a context change.

Building Personalized Mobile Phone Interruption Models. Rosenthal et al. (Rosenthal et al., 2011) proposed a method to learn when to turn off and on the phone volume to avoid phone interruptions. Specifically, they presented a phone volume application that classifies users' interruptability and adjusts the volume accordingly. They introduced an experience sampling technique that asked users to predict their costs of interruption and used these predictions to approximate a cost model and determine when to actually ask for preferences.

They used the interruptibility preference data collected via experience sampling to build a classifier to determine when users want their phone to ring (i.e., when they are interruptible).

In this research direction, Apple has recently introduced the *don't disturb* feature on iOS 6 allowing users to schedule the periods of time when the phone has to be silenced. The fact that a company leader in the mobile devices field emphasizes the interruption problem and tries to make a first attempt to solve it is an indicator that more research is needed.

3.4.1 Analysis and discussion

Research in the non-intrusive computing area has been mainly focused on reducing the perceived burden of interruptions. However, the different approaches are almost exclusively focused on evaluating the adequate timing for interruptions (i.e., the *right* time), while user interface adaptation or presentation mode has received few attention. Conversely, our approach analyzes the current context of the user and adapts the interaction in terms of obtrusiveness according to it (i.e., the *right* way).

Table 3.3 and Table 3.4 shows the compared dimensions of the different proposals presented in this section. Most of the approaches presented provide support to user preferences to adapt the system according to each user. The description of the system's adaptive behavior is carried out in a different way by using scenario-based descriptions, state diagrams, a set of known candidates or fuzzy sets. Some approaches (MIPCE and Bayesphone) do not explicitly specify the variability of the system since they support the variability by making inferences. Also, most of the approaches support the obtrusiveness by means of intrusion or interruptibility models that generally model the cost of interrupting the user. Conversely, our approach models the obtrusiveness by taking into account the initiative and attention level of users.

None of the approaches provide tools to describe the system (see *tool support* row). Moreover, they are not supported by a well-defined methodology to define the system. In some approaches, the user participates in the system specification implicitly (see *user participation* row) by means of the training period; thereby, these systems require a training period (see *training* row).

All the approaches that provide adaptation capabilities possess an adaptive behavior. However, none of the approaches support multimodality (see *multimod. support* row) because they are focused on evaluating when interrupting the user. Conversely, our approach is focused on how interrupting the user according to context. Regarding

Property	**TMSMP**	**SenSay**	**MIPCE**	**Bayesphone**
User pref. support	no	yes	yes	yes
Adap. behav. modeling	scenarios description	state diagram	no	no
Obtrusiv. of adaptation	level of intrusiveness	interruptability states	intrusion level	interruptability models
Tool support	no	no	no	no
Well-defined methodology	no	no	no	no
User participation	no	no	no	yes
Flexibility in adaptation	adaptive	adaptive	adaptive	adaptive
Multimod. support	no support	no support	no support	no support
Integration capabilities	no	no	users & devices	users & devices
Adaptation rules	hardcoded	transition rules	negotiation algorithm	cost-benefit policies
Runtime models	no	no	no	no
Training	no	no	no	yes
Customizab.	no	no	system	system
Personaliz. ind. needs	no	manually	manually	manually
Open adaptation	no	no	no	no
Domain	phone calls	phone calls	meeting environment	phone calls

Table 3.3: Related work from the non-intrusive ubiquitous computing perspective

the integration capabilities, most of the approaches support a consistent view of users, however, only three of them support different devices.

These proposals offer different mechanisms for describing the adaptation rules: hardcoding the rules at code-level, defining transition rules

Property	**RPBI**	**QANIC**	**CAMPP**	**BPMPIM**
User pref. support	yes	yes	yes	yes
Adap. behav. modeling	-	known candidates	fuzzy sets	-
Obtrusiv. of adaptation	interruption level	intrusion level	no	interruptibility cost
Tool support	no	no	no	no
Well-defined methodology	no	no	no	no
User participation	yes	no	no	yes
Flexibility in adaptation	-	adaptive	adaptive	-
Multimod. support	no support	no support	no support	no support
Integration capabilities	users	users & devices	users	users
Adaptation rules	-	comparison model	IF-THEN rules	-
Runtime models	no	no	no	no
Training	yes	no	no	yes
Customizab.	system	no	system	system
Personaliz. indiv. needs	no	no	manually & automatically	automatically
Open adaptation	no	no	no	no
Domain	proactive messages	IM system	phone profiles	phone calls, SMS

Table 3.4: Related work from the non-intrusive ubiquitous computing perspective

in the state diagram, providing a negotiation algorithm to determine the most appropriate device, using cost-benefit policies, using a comparison model, or by means of IF-THEN rules. However, none of them use these models at runtime to provide the adaptation of the system.

Finally, some of the approaches use learning techniques to adapt the system. However, the control of the learning is made only by the system, without letting the user to have a control over the system. Conversely, our approach allows users to change the behavior of the system by means of end-user interfaces that update the design models. Also, only CAMPP and BPMPIM learn the new user preferences automatically by means of the user's context and training data. However, none of the approaches support unknown environments.

It is worth noticing that all the application domains of the approaches are limited to phone calls or SMS messages. Our approach integrates different ubiquitous services to give them interaction adaptation capabilities in terms of obtrusiveness.

3.5 Discussion of previous systems

Previous systems spend much effort on exploring various aspects of interaction adaptation and intrusiveness of ubiquitous systems. However, they address the two aspects independently. Also, service integration is not often considered. Hence, existing systems are mostly limited to a particular application domain. Most of these approaches also lack of a well-defined methodology that guides designers in the system specification, and they are not supported by tools. This makes the systems difficult to design, evolve and maintain. They support user preferences, however, few of them allow the user participation in the system specification. We consider these missing points for defining our approach. In the following section, we describe the characteristics of our proposal.

3.5.1 Characteristics of our proposal

Based on the lacks of the related approaches, our proposal provides the following characteristics:

Introduction of the obtrusiveness aspects in the interaction adaptation. A distinguishing aspect of our work is the consideration of the interaction obtrusiveness as a first-class concept. We introduce obtrusiveness requirements in the adaptation process, while other approaches focused on modeling context-aware user interfaces do not consider this factor.

User needs drive the adaptation. This work follows a user-centered design method in which user needs drive the design of the system providing users with personalized services and avoiding information overload. In this way, we enhance the user experience by adjusting the interaction of services according to the attentional resources of each user.

A well-defined interaction obtrusiveness adaptation technologically independent. Most of the research in Attentive User Interfaces present theoretical frameworks for designing attentive notification systems; however, they lack methodological support and guidance to develop this kind of systems. This work provides a set of well-defined steps in order to develop interaction obtrusiveness adaptation and to make this attentive notification vision a reality.

By means of using feature models, we define (1) the commonalities and differences of the different contexts in terms of the interaction features that each context supports, and (2) the constraints for their selection allowing a multimodal interaction. Moreover, as models are technology-independent, we can adapt the interaction of the various mobile devices that a user can possess, regardless of the platform.

A runtime infrastructure to achieve the dynamic adaptation. Previous research in non-intrusive computing is focused on minimizing interruptions that generally work over a limited set of message sources and one delivery channel at a time. We propose an infrastructure that aims to give the capability of interaction adaptation to the different pervasive and mobile services, allowing different interaction components to be composed, i.e., it is not limited to one interaction mechanism at a time. In addition, as we use the design models at runtime, interaction adaptation is performed from the first moment the system is running, avoiding having to have training data and cold-starts.

A learning system to adjust the a-priori designed adaptations. Some of the presented approaches relies on the designer knowledge to capture all the situations that influence the adaptation of systems for all users. This can lead to innacuracy and ambiguity in the adaptations. Another approaches that learn automatically how to adapt the system, do not allow users to have a control over the system. Our approach addresses these two challenges by providing a learning system capable of re-adjusting at runtime the initial models specified at design time. Furthermore, we provide customization interfaces to let the user change the behavior of the system and to have control over adaptations.

3.6 Conclusions

This chapter presents the state of the art in the disciplines that are related to this work. These areas are really active these days with many initiatives. However, there is a lack of proposals to provide mechanisms that allow the development of services' interaction obtrusiveness adaptation within the three application domains. Modeling languages are used only to describe context-aware user interfaces which do not take into account user attention. Towards creating systems that adapt their level of intrusiveness to the context of use, researchers and designers face a design challenge in terms of creating devices that sense user's state and attention in order to calculate the better time for interruptions.

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

The insufficiencies in the previous systems guide the design and construction of our interaction obtrusiveness adaptation approach. The next chapter gives an overview of our proposed approach.

4 Overview of the Proposal

The big picture

Everything has been composed, just not yet written down.
—*Letter to Leopold Mozart (1780).*

With the advanced capabilities of ubiquitous devices (e.g., mobile phones, tablets, TVs) such as connectivity, positioning systems, sensors, advanced interaction mechanisms, new valuable services can be provided. For example, customers in a supermarket can be informed about meaningful information such as special offers, friend's opinions about the product or they can even compare prices of products directly interacting with the physical products. As a result of these innovations, the way users interact with services is changing (Schmidt et al., 2012). Traditionally, users in their desktop environments are used to explicitly request for information or input data. Today's ubiquitous access to services embedded in the user's environment enables the use of implicit interactions (Ju and Leifer, 2008); those that occur without the explicit behest or awareness of the user. For example, a user can return a book in a library by simply leaving the book in the return box.

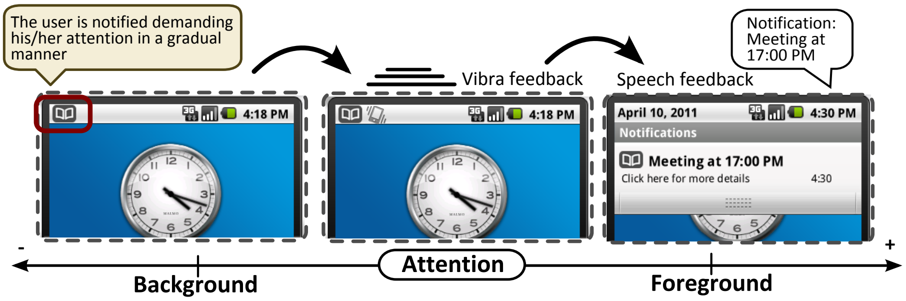


Figure 4.1: Example scenario where the attention demand is increased in a gradual manner for a mobile phone.

Furthermore, ubiquitous devices accompany their users throughout the entire day and many interactions occur when users are actively engaged in other tasks, resulting in unwanted interruptions (Gibbs, 2005). Thus, systems should be able of dynamically adapting interactions in terms of obtrusiveness to the current context of the user in order not to interrupt him/her. For example, as illustrated in Fig. 4.1, a user can be notified in a gradual manner, instead of immediately interrupt his/her ongoing activity. For achieving this, the attention demand can be increased as the deadline of a notification approaches: first a non-intrusive mark on the screen can be used to indicate that a notification exists, then a soft vibration can be delivered, and finally, auditory feedback (speech or sound) is produced if the user is not still aware of it. These design and runtime requirements introduce complexity in the development of ubiquitous services.

This thesis faces this development of *interaction obtrusiveness adaptation* in ubiquitous and mobile contexts by considering their analysis, design, development, and runtime adaptation. The idea of the approach that we propose in this thesis is to combine model-driven techniques and the core issues of interaction design to provide ubiquitous devices with user-centered self-adaptive capabilities in terms of obtrusiveness. Firstly, models are used to centralize the knowledge of the system and capture the rationale behind it. Models cope with complexity through abstractions and are used both to specify the dynamic interaction variability and to drive runtime interaction adaptations. Secondly, the principles of interaction design are used to manage user attention requirements (i.e., obtrusiveness aspects) and adaptation concerns in terms of obtrusiveness. In this way, the design is driven by human needs with the goal of achieving a better user experience.

The remainder of this chapter provides a high-level view of the building blocks involved in our approach, and Chapters 5-7 provide more detailed discussion of the methods involved in each of these building blocks. Specifically, this chapter is structured as follows. Section 4.1 introduces the types of systems we address in this work. In Section 4.2, we identify and overview the main building blocks of the approach and present the process to apply this approach. In Section 4.3, we introduce the systems' infrastructure to leverage the design models at runtime. Then, we show how the approach has been put into practice and evaluated by means of a case study-based evaluation in Section 4.4. Finally, Section 4.5 concludes the chapter.

4.1 Point of view

According to the canonical models of human-computer interaction, the information flows from a system through a person and back through the system again based on a feedback loop. Within this feedback loop, Dubberly et al. (Dubberly et al., 2009) have distinguished the types of interactive systems based on its degree of interaction with the user. Specifically, the following types of systems can be provided (see Fig. 4.2):

Linear system. These systems are those that only react to input, for example, when the supermarket door opens automatically as you step in front of it. The design of these systems follows the input-output pattern, it means, the output function is fixed.

Self-regulating system. These systems are a specific type of closed-loop systems (single loop) that have a goal to regulate. The goal

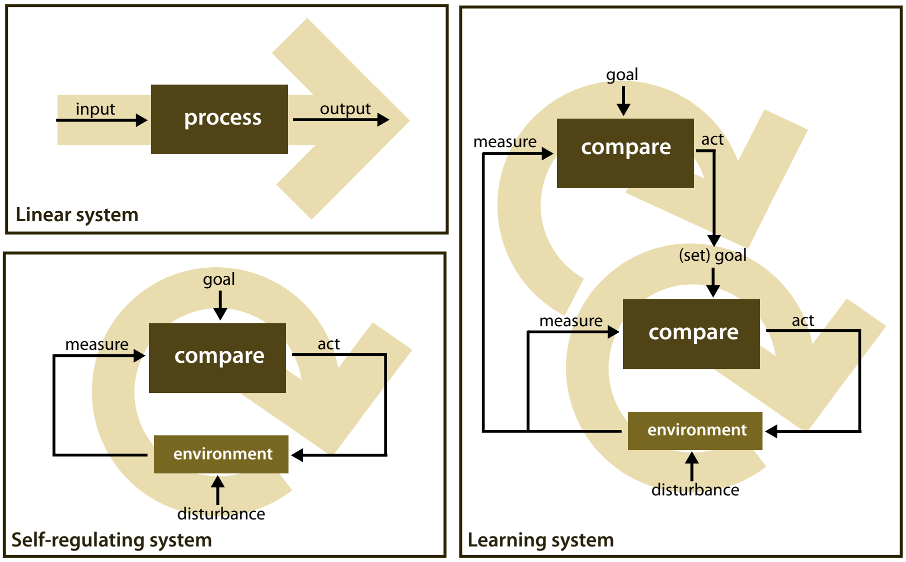


Figure 4.2: Types of interactive systems.

defines a relationship between the system and its environment, which the system seeks to attain and maintain. A simple self-regulating system cannot adjust its designed goal; its goal can be adjusted only by something external to the system.

Learning system. These systems nest a first self-regulating system inside a second self-regulating system, where the second system measures the effect of the first system on the environment and adjusts the first system's goal according to how well its own goal is being met. The second system adjusts the goal of the first based on external action.

We consider all these degrees of interactivity in this thesis. First, a *linear system* is proposed when we design a fixed degree of obtrusiveness for the interaction with a specific service according to each user's needs and preferences. This is what we call **personalization** in terms of obtrusiveness. Second, a *self-regulating system* is provided when the system dynamically regulates the degree of obtrusiveness required for a service according to the user's situational context. This is what we call **self-adaptation** in terms of obtrusiveness: a system's capability to gather information about user's current situation, to evaluate this information and to change its obtrusiveness behavior according to the current situation. Third, a *learning system* is proposed to automatically adjust the self-regulating system design according to the changing user needs and preferences over time. This is what we call **self-adjusting** for improving the obtrusiveness design.

To represent the knowledge of each one of these systems, we based them on **models**. Models are used as powerful tools for interaction design in order to represent complex phenomena with a useful abstraction (Cooper et al., 2007). They organize the knowledge required for the behavior of our services by means of the explicit modeling of interaction variability and obtrusiveness dependencies. In this way, designers can focus on these requirements at a high level of abstraction, deferring technological decisions and implementation issues. In addition, these models are created following a user-centered design (Mao et al., 2001) in which user needs drive their design. This allows us to design for users.

Also, the self-adaptation and self-adjusting capabilities of interactive systems require the system to infer knowledge from the current situation to trigger an appropriate response (self-adaptation) and change the designed models during runtime (self-adjusting). Following the **Autonomic Computing** principles (Kephart and Chess, 2003), the system makes decisions on its own, using high-level policies; it constantly check and optimize its status and automatically adapt itself to changing conditions. To achieve this autonomic behavior, we leverage the models produced at design time as if they were the high-level policies that drive the autonomic behavior of the system at runtime. Runtime models have been proven to provide a richer semantic base for runtime decision-making in order to achieve system adaptation, since they provide up-to-date and exact information about the runtime system (France and Rumpe, 2007).

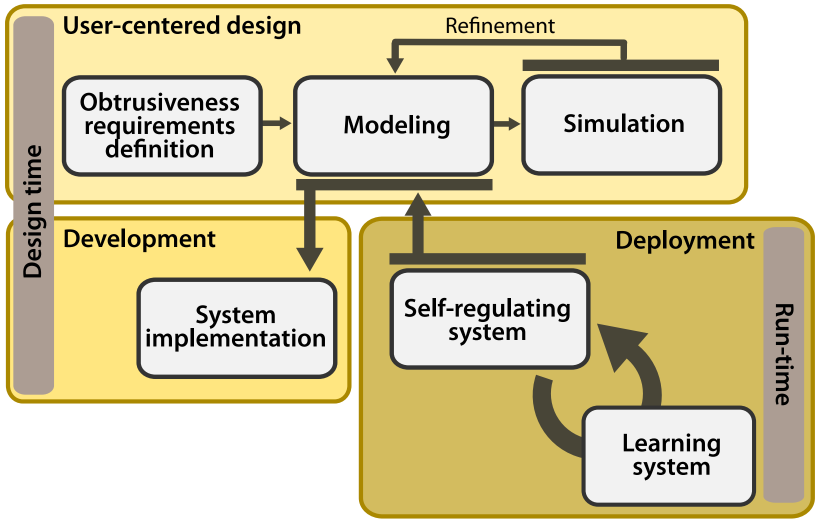


Figure 4.3: Main building blocks of our approach.

4.2 Main building blocks

Figure 4.3 shows the overall view of the proposed approach decomposed in the main building blocks. The framework can be considered as the collection of two subsystems: from a methodological perspective the approach is divided in two phases; design time and runtime. At design time, the user obtrusiveness requirements are defined and the models that specify the interaction obtrusiveness behavior are built. Then, the designed obtrusiveness is put into practice to obtain feedback from the users and refine the models. When no further changes are required, the system specification is used to guide the development of the final system. At runtime, following the self-regulating strategy, these models are queried in response to context events to produce the adaptation of the interaction in terms of obtrusiveness. This adaptation is adjusted to the changing user preferences over time by means of the learning strategy. The main building blocks of the approach, the steps involved

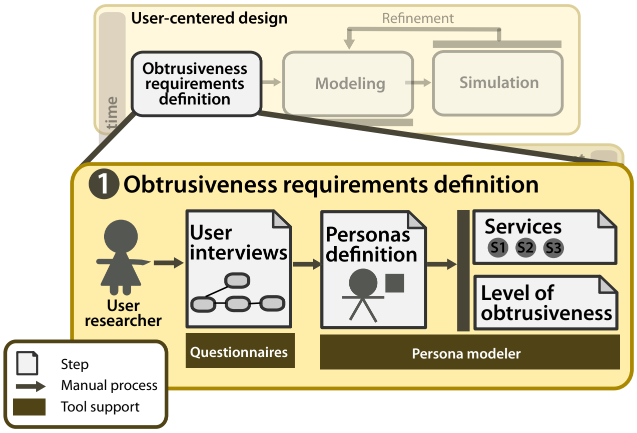


Figure 4.4: Steps of the obtrusiveness requirements definition phase.

in each one of them and the tool support are the following.

Obtrusiveness requirements definition. 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 (user researcher) 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*** (Brown, 2010) (user profiles). Specifically, *personas* are user archetypes used to represent the relevant information of the audience such as behaviors, attitudes, goals, motivations, etc. identified during the interviews and observations. 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 of the system and their level of obtrusiveness*** according to the user context. By establishing the degree of user attention that a service needs, we avoid developing overwhelm-

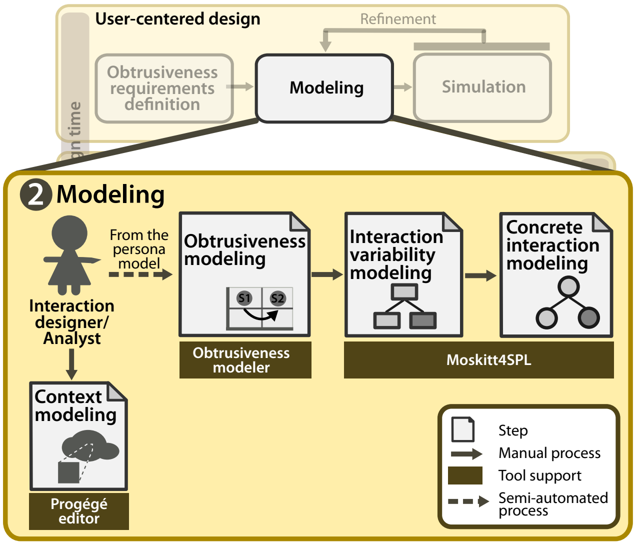


Figure 4.5: Steps of the modeling phase.

ing services. These concepts are expressed together in the models created in the next stage. Figure 4.4 illustrates the process to be followed to define the obtrusiveness requirements. More detail regarding the analysis and specification of these concepts is provided in Chapter 5.

Tool support. To make the user interviews, designers follow a set of *questionnaires*. Then, the *Persona Modeler* is used to define the personas and specify the services required and their obtrusiveness level.

Modeling. Once the user requirements are captured, the different models that characterize interaction obtrusiveness adaptation are defined and the mappings among these models are specified in this stage. Specifically, designers ***model the obtrusiveness*** by defining an *unobtrusive adaptation space* to capture the attentional demand required for each service. In this model, the transitions that define how the obtrusiveness of a service can change at runtime are also defined. These transitions link user situations (composed by context events) with changes in the obtrusiveness. In order to formalize the context and capture the state of the user's environment to trigger transitions, analysts ***model the context***.

Also, to ***model the interaction variability*** according to each degree of obtrusiveness, we use *feature models*. This allows interaction designers to describe the commonalities and variabilities between interaction modalities available that are going to give support to the different levels of obtrusiveness defined. As feature models are used to model interaction in an abstract manner, designers have to define the concrete interaction components of the target platform that support the interaction modalities defined by features. This is done by using a *concrete interaction model* in the ***concrete interaction modeling*** step. Figure 4.5 illustrates the process to be followed to define the models of the modeling phase. More detail regarding the specification of these models is provided in Chapter 5.

Tool support. 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 *Protégé-OWL* editor is used for modeling the context in a graphical manner. Finally, *MOSKitt4SPL*, a free open-source tool, is used to create the feature model, the concrete interaction model and the mappings between them.

Simulation. Once designers consider that the behavior defined for adaptation is the one desired, several iterations with end-users are performed in order to improve and refine the obtrusiveness design. Thus, before efforts are put into development, fast-prototyping is applied to validate the user satisfaction with the interaction obtrusiveness adaptation defined for services. First, the design team ***define validation scenarios*** in order to illustrate the way interaction is adapted in terms of obtrusiveness. According to these scenarios, designers ***define the screen mock-ups*** needed to sup-

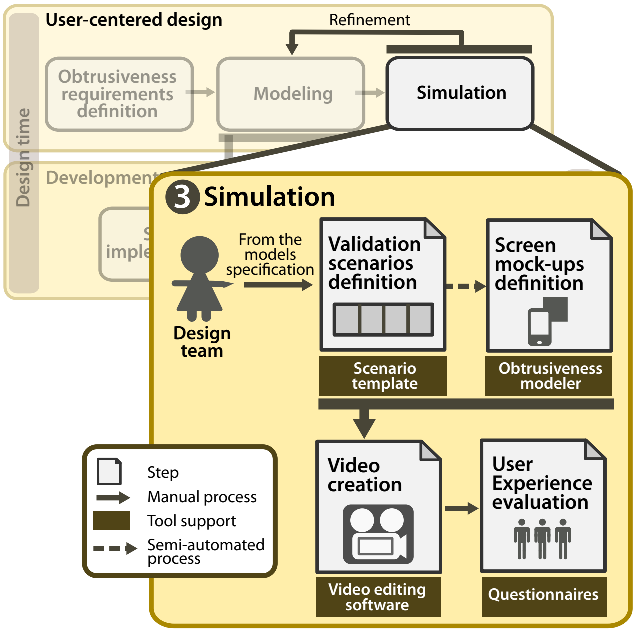


Figure 4.6: Steps of the simulation phase.

port the different services in the obtrusiveness levels. These mock-ups can be obtained automatically from the models defined.

Then, the design team define *rapid video prototypes* (Kuniavsky, 2010) to ***create conceptual videos*** of the proposed obtrusiveness adaptation. With these videos, end-users are capable to see enough details of the considerate interaction adaptation of services according to each situation. They provide a quick exploration of the user experience by using our system. Also, we use *Wizard-of-Oz prototyping* (Dahlbäck et al., 1993) to simulate the interaction adaptation effect on users and ***evaluate the user experience***. In this way, the user is immersed in an environment that behaves like a working system with obtrusiveness adaptation capabilities, but it is much easier to produce. Finally, once designers obtain the feedback from users, they ***refine the models*** to better fit

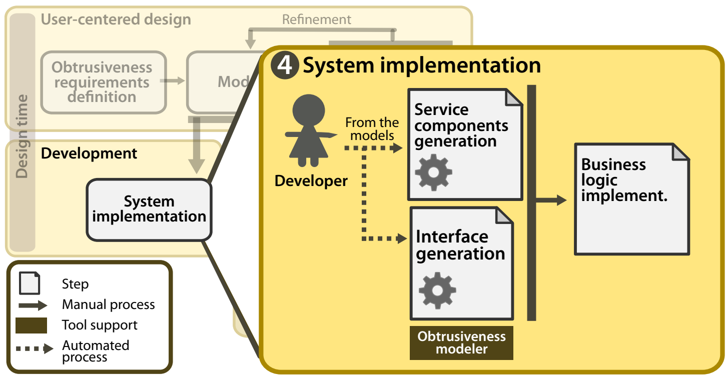


Figure 4.7: Steps of the implementation phase.

with the user needs and preferences. Figure 4.6 illustrates the process to be followed to simulate the system design. Chapter 5 provides the guidelines to perform the simulation.

Tool support. *Scenario templates* are used to define the validation scenarios. Based on these scenarios and using the *Obtrusiveness Modeler*, the different screen mock-ups to support each interaction can be obtained automatically from the models defined. Then, for the video creation any *editing software* can be used. Finally, the user experience is evaluated through user satisfaction *questionnaires*.

System implementation. Once the models have been adjusted to fit with the real user needs, a final adaptive software solution can be obtained. Following the Model-Driven Engineering development, the derivation of a software solution from the system specification is produced automatically. From the models obtained in the modeling stage, developers can ***generate the service components*** (in conjunction with service architectural models) and ***generate the different variants of the user interfaces*** to support the interaction with the services. In particular, we make use of code generation techniques in this work to automate the development of the user interfaces for different target technologies.

Then, developers should implement the business logic of each service taking into account several considerations in the interaction layer of the services to enable the adaptation capabilities. These considerations are related to the functionality to be adapted and the infrastructure required for the communication between our self-regulating system and the services. Figure 4.7 illustrates the process to be followed to implement the system.The derivation of a software solution is described in Chapter 6.

Tool support. The service components and the user interfaces can be generated automatically using the *Obtrusiveness Modeler* by means of code generation capabilities.

Self-regulating system. The design models capture the knowledge required to provide service interactions with the appropriate obtrusiveness. However, in order to allow an autonomic and adaptive considerate behavior, the system must itself regulate the interaction provided for each service according to the different user situations. Since the adaptation in our approach is driven by models at runtime, a *model-based autonomic infrastructure* is provided to support a self-adaptive interaction obtrusiveness of services in the ubiquitous computing domain. This infrastructure exploits the design models at runtime to determine how the interaction should be adapted in terms of obtrusiveness. Thus, to ***deploy the self-regulating system***, analysts have to save the models into the infrastructure folder and run its components. This self-regulating system is further explained in Chapter 6.

Tool support. To deploy the self-regulating system, analysts use *AdaptIO featuring a deployment strategy.*

Learning system. As the self-regulating system is based on an obtrusiveness behavior defined at design time, these decisions made at design time may change over time since user needs and preferences can change. In order to adjust these decisions, we provide a *learning system.* This system is in charge of re-adjusting the obtrusiveness of services according to users' behavior automatically at runtime. To do this, it follows a reinforcement learning approach for self-adjusting the designed interaction obtrusiveness according to user's feedback, in a way that maximizes user's satisfaction for a long-term use. To ***deploy the learning system***, analysts have to run its components. In addition, we also provide *customization interfaces* to allow users to set up his/her new obtrusiveness preferences explicitly, as they are who better know his/her preferences. Further detail regarding the learning strategy is provided in Chapter 7.

Tool support. To deploy the learning system, analysts use *AdaptIO featuring a deployment strategy.*

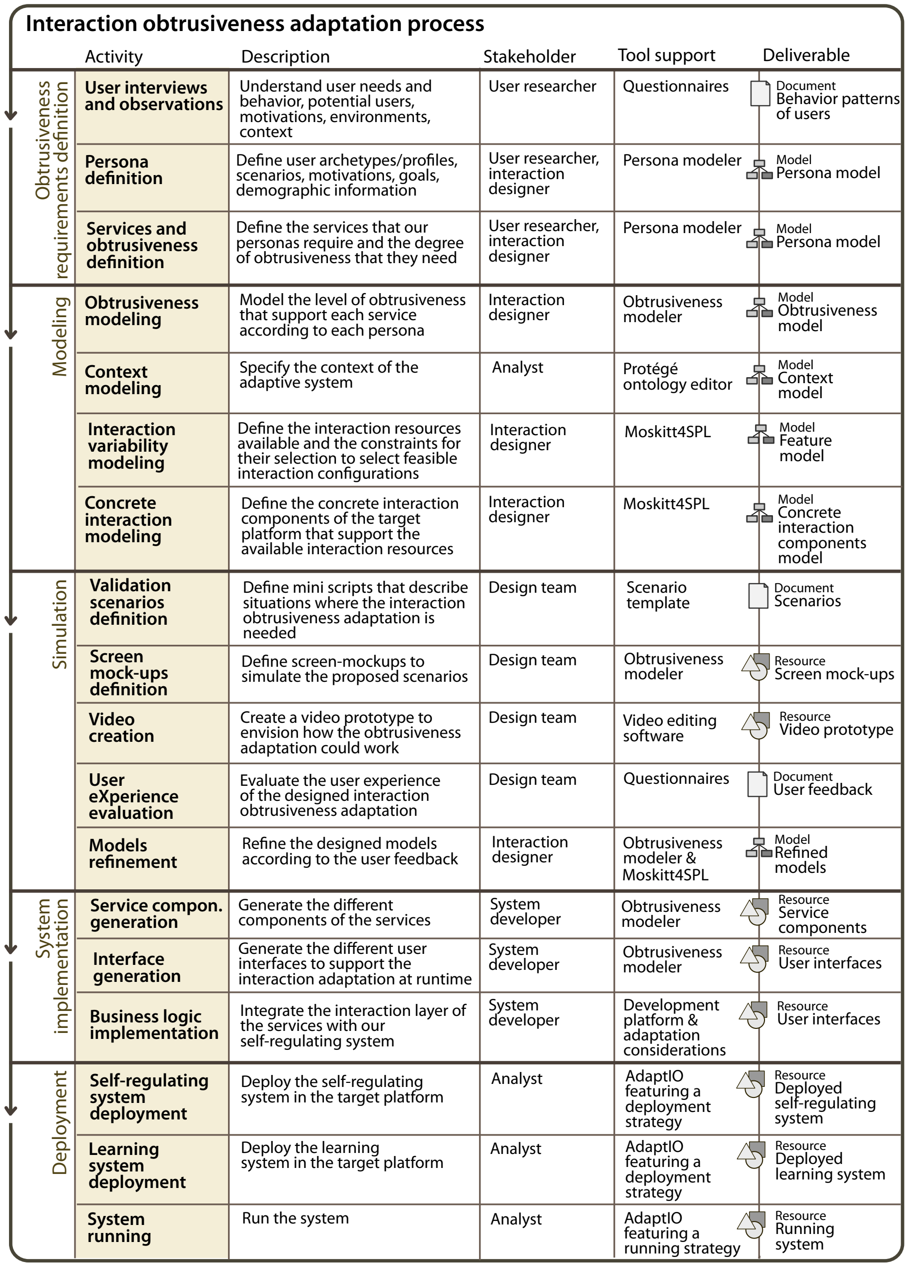
Figure 4.8 shows a more detailed diagram of the process, including the name of the activity, a description, the stakeholder involved, the tool support and the deliverables produced.

The above building blocks constitute the **AdaptIO** framework, with the goal of facilitating the building of ubiquitous systems capable of adapting its interaction obtrusiveness according to user's context. With this building blocks we provide support to interaction designers and developers from system design to execution.

It is worth noticing that the steps of the modeling phase do not need to be performed by the designer strictly sequentially. Models can be edited during the whole design process. Further detail regarding the steps of the obtrusiveness requirements definition, the modeling, and the simulation phases is provided in Chapter 5. The system implementation phase is described in Chapter 6 together with the self-regulating system and its deployment. Finally, the learning system is detailed in Chapter 7 and tool support is described in Appendix A.

4.3 Systems' infrastructure

To support the interaction obtrusiveness adaptation by following the process explained in Section 4.2, we provide two systems that leverage the design models at runtime at different degree: *interpretation* and *modification*.



Interaction obtrusiveness adaptation process

	Activity	Description	Stakeholder	Tool support	Deliverable
Obtrusiveness requirements definition	**User interviews and observations**	Understand user needs and behavior, potential users, motivations, environments, context	User researcher	Questionnaires	Document Behavior patterns of users
	Persona definition	Define user archetypes/profiles, scenarios, motivations, goals, demographic information	User researcher, interaction designer	Persona modeler	Model Persona model
	Services and obtrusiveness definition	Define the services that our personas require and the degree of obtrusiveness that they need	User researcher, interaction designer	Persona modeler	Model Persona model
Modeling	**Obtrusiveness modeling**	Model the level of obtrusiveness that support each service according to each persona	Interaction designer	Obtrusiveness modeler	Model Obtrusiveness model
	Context modeling	Specify the context of the adaptive system	Analyst	Protégé ontology editor	Model Context model
	Interaction variability modeling	Define the interaction resources available and the constraints for their selection to select feasible interaction configurations	Interaction designer	Moskitt4SPL	Model Feature model
	Concrete interaction modeling	Define the concrete interaction components of the target platform that support the available interaction resources	Interaction designer	Moskitt4SPL	Model Concrete interaction components model
Simulation	**Validation scenarios definition**	Define mini scripts that describe situations where the interaction obtrusiveness adaptation is needed	Design team	Scenario template	Document Scenarios
	Screen mock-ups definition	Define screen-mockups to simulate the proposed scenarios	Design team	Obtrusiveness modeler	Resource Screen mock-ups
	Video creation	Create a video prototype to envision how the obtrusiveness adaptation could work	Design team	Video editing software	Resource Video prototype
	User eXperience evaluation	Evaluate the user experience of the designed interaction obtrusiveness adaptation	Design team	Questionnaires	Document User feedback
	Models refinement	Refine the designed models according to the user feedback	Interaction designer	Obtrusiveness modeler & Moskitt4SPL	Model Refined models
System implementation	**Service compon. generation**	Generate the different components of the services	System developer	Obtrusiveness modeler	Resource Service components
	Interface generation	Generate the different user interfaces to support the interaction adaptation at runtime	System developer	Obtrusiveness modeler	Resource User interfaces
	Business logic implementation	Integrate the interaction layer of the services with our self-regulating system	System developer	Development platform & adaptation considerations	Resource User interfaces
Deployment	**Self-regulating system deployment**	Deploy the self-regulating system in the target platform	Analyst	AdaptIO featuring a deployment strategy	Resource Deployed self-regulating system
	Learning system deployment	Deploy the learning system in the target platform	Analyst	AdaptIO featuring a deployment strategy	Resource Deployed learning system
	System running	Run the system	Analyst	AdaptIO featuring a running strategy	Resource Running system

Figure 4.8: A more detailed look at the process to apply our approach.

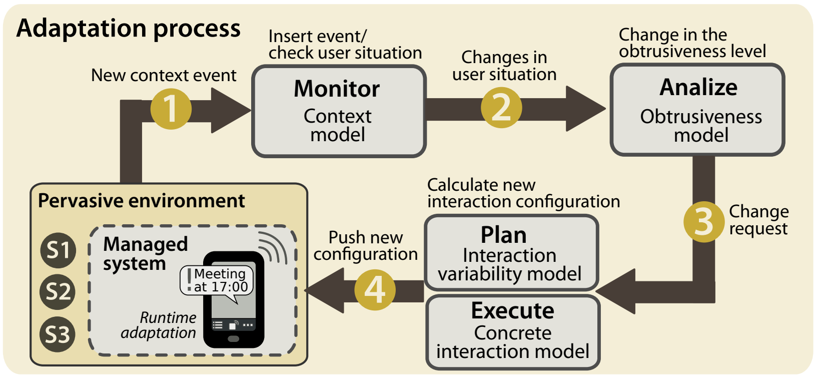


Figure 4.9: Overview of the adaptation process.

At the interpretation level, the models are queried at runtime in order to calculate the interaction obtrusiveness adaptation according to context changes. At the modification level, the models are updated in order to adjust them to the new user needs and preferences over time. As the models are exploited at runtime, changes in them do not require an explicit regeneration, rebuildment, retestment, and redeployment of the system[1]. Specifically, the provided systems are the following:

Self-regulating system. The design models capture the knowledge required to provide interactions with the appropriate obtrusiveness for each situation. In order to make our user-centered obtrusiveness interaction adaptation a reality, we define a *model-based autonomic infrastructure*. This infrastructure exploits the design models at runtime to support self-adaptive interaction obtrusiveness of services in the ubiquitous computing domain. Specifically, our infrastructure is based on the IBM reference model for self-management (IBM, 2006), which is called MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge).

The overall adaptation steps are outlined in Figure 4.9. When a

[1]http://www.theenterprisearchitect.eu/archive/2010/06/28/model-driven-development-code-generation-or-model-interpretation

context monitor senses a new context event, it inserts the change in the context model and checks if the user situation has changed and some adaptation needs to be made. If so, the obtrusiveness model is checked to see if any obtrusiveness adaptation has to be triggered according to the new user situation. A change request is generated with the new obtrusiveness level. With this change request, the necessary modifications to the interaction resources are calculated in terms of increments and decrements with respect to the last obtrusiveness configuration. Also, the new interaction configuration is translated to the concrete interaction components of the underlying managed system. Finally, the infrastructure pushes the new interaction configuration to the managed system in order to apply the adaptation. More detail about this infrastructure and its implementation is covered in Chapter 6.

Learning system. In order to improve the initial obtrusiveness design and adapt the obtrusiveness degree for each service to the new user preferences, we follow a Reinforcement Learning (RL) strategy (Sutton and Barto, 1998a). Following this strategy, our infrastructure learns from user feedback after interacting with a service in an obtrusiveness level. Depending on the received feedback (positive or negative), the current obtrusiveness level of a service is rewarded or punished. In this way, if negative feedback is received continuously for a given service in an obtrusiveness level, it means that the user is unsatisfied with the design and the system will adjust it by changing the obtrusiveness level. This obtrusiveness adjustment is achieved automatically (self-adjustment). However, we also let users to explicitly set up his/her new preferences by means of two customization interfaces:

- *Obtrusiveness Personalization Interface.* To allow users define their own obtrusiveness preferences, we provide an obtrusiveness personalization interface. In this way, the obtrusiveness behavior is adapted to each user without any training. In particular, we provide a personalization interface to let users change the unobtrusive adaptation space and the linking with the interaction resources in a user-friendly interface.

- *User Situation Specification Interface.* In order to guarantee accurate and unambiguous user situation definitions, the user is put in charge of defining his own situations by means of a situation specification interface. This interface allows users to expressively define and modify their situations in a generic and fine-grained way, based on environment context. To increase usability and support nomadic users in a wide range of environments, the interface also supports directly capturing user situations.

Further detail regarding the learning system and the different customization interfaces is provided in Chapter 7.

4.4 Evaluation of our approach

The presented work has been validated from different perspectives according to the confronted research questions. These validations are the following:

1. User-centered design method.

- **Usability of our design method.** The introduced user-centered design method has to be expressive enough to describe the aspects that are relevant to capture the interaction obtrusiveness adaptation requirements. Also, this design method has to help designers in the specification of this kind of systems, from requirements to deployment. Therefore, we have conducted an experiment to evaluate the usability of our design method compared to the traditional method (hand-coding).
- **User satisfaction with the designed adaptations and workload.** As the users are who are going to use the system, it has to be accepted by them and designed according

to their needs and preferences. Thus, before system implementation, we conducted a user satisfaction evaluation to determine whether the designed interaction adaptations in terms of obtrusiveness were well accepted by the users and fit all their needs. Also, we conducted a cognitive load study to analyze how our system is capable of handling the attention resources of the user by means of the adaptations. This evaluation is part of the simulation phase of our method.

2. Self-regulating autonomic infrastructure.

    - **Scalability of the self-regulating infrastructure.** The provided infrastructure has to be subject to the same efficiency requirements as the rest of the system because the execution of the adaptation impacts the overall system performance. Therefore, we analyzed the scalability of our infrastructure by studying the temporal cost of the operations that access the models. In particular, we have evaluated the performance of manipulating the models at runtime.
    - **User experience with the system.** One of the goals of our approach is to enhance the User eXperience (UX) by giving ubiquitous services the capability of adapting its interaction to the attentional resources of users. Therefore, we evaluated the User eXperience after users interacted with our system (to evaluate their feelings). For this evaluation, we provided to users with a non-adaptive version of our system and the adaptive one. In this way, users could compare both systems and better measure their UX.

3. Obtrusiveness learning system.

    - **Efficiency of the learning system.** The learning system has to be capable of adjusting the behavior of the system when the user changes his/her preferences or needs (a change in his/her mind). This learning should be quick enough and consistent with the user's mind in order to guarantee user satisfaction with the system. Thus, we have measured the efficiency of the learning system by analyzing the behavior of it when the user changes his/her mind.

- **Usability and expressivity of the Customization Interfaces.** The mechanisms and tools provided for defining user situations and customizing the obtrusiveness must allow end-users to define and update them with enough usability and expressivity. Therefore, we have validated the usability and expressivity of these interfaces by means of a user evaluation, where users had to interact with the interfaces for defining and changing different situations and obtrusiveness adaptations.

To evaluate the above concerns, we have carried out a case study based evaluation. To perform it, we have developed a Smart Home case study and an Adaptive Notifications case study (see Chapter 8) following the guidelines for case study research by Runeson and Höst (Runeson and Höst, 2009). The Smart Home adapts the obtrusiveness of its services according to changes in the user's context and the Adaptive Notifications adjusts the interaction of the notifications delivered by different ubiquitous services according to the current user situation.

Overall, the evaluation of the case studies revealed positive results that can encourage designers and developers to apply our approach to ubiquitous services. Regarding the user-centered design method, the evaluation points out the usability and usefulness for system designers and the user satisfaction with the interaction obtrusiveness adaptations that can be designed; nevertheless, the participants' feedback in the last study revealed that several iterations are needed to maximize user satisfaction with the designed obtrusiveness adaptation. The evaluations about the self-regulating system also achieved satisfactory results regarding the scalability of the system, and the user experience was enhanced with the adaptive system. Finally, high efficiency was obtained regarding the learning system. Also, the evaluations about the interfaces showed them to be usable and expressive, allowing users to specify non-trivial situations and adaptations within a short time span.

However, during the experiments, we found that *intelligibility* can become an issue that affect user satisfaction and experience, as the adaptation is transparently to users and they can feel loss of control. This can also cause uncertainty of what the system is going to do or has learned. Offering the users the possibility to check adaptations could reduce this uncertainty and lack of control. Thus, this aspect needs to be improved to give users more control over their systems.

4.5 Conclusions

The challenge in an environment full of embedded services (where human attention is the most valuable resource) is not only to make information available to people at any time, at any place, and in any form, but to reduce information overload by making information relevant to the task-at-hand (Fischer, 2001). Information delivery methods should achieve the right balance between the costs of intrusive interruptions and the loss of context-sensitivity of deferred alerts (Horvitz et al., 2003).

In this thesis, we manage user's attentional resources in order to make ubiquitous interactions less obnoxious. To achieve this, we propose an approach to design and develop ubiquitous services capable of adjusting its interaction obtrusiveness dynamically according to the user's context. As the whole method is supported by models, feedback from users is easily mapped onto the models. Also, the use of models to design the interaction obtrusiveness adaptation is useful to centralize the knowledge and organize it in a way that it is easy to handle for designers (e.g., work with technology-independent concepts, detect inconsistencies, etc.). Also, the models support the evolution of the services to new technologies and platforms.

The definition of our autonomic infrastructure allows the services to be self-adaptive to the different user situations in order to regulate their interaction obtrusiveness. Also, by means of the learning system, the interaction adaptation behavior of the services is re-adjusted automatically according to the user's feedback. Finally, by means of applying our method in two case studies, we show that by following our approach, personalized services with the properly interaction in terms of obtrusiveness can be obtained.

5

A Framework for Interaction Obtrusiveness Adaptation

Designing considerate interactions

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

—Ben Shneiderman, 2003.

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

An important challenge in the development of ubiquitous services is the need for adapting interaction to the user, taking into account not only his/her current context, tasks and priorities, but also the user's level of attention according to his/her needs (Piva et al., 2005). Users need to receive different feedbacks from services and interact in a different manner according to all these aspects. As an example, consider the different interaction styles between the same user going in a hurry to take a fly and watching relaxed the TV.

This chapter introduces a methodological approach and a framework for the design of considerate interaction adaptations in the ubiquitous domain. The goal of this method is to provide a mechanism for **defining the desired obtrusiveness variability of services' interactions according to user's attention**. In order to systematize the development of such services and achieve a well-designed system according to each user, the method is based on the user-centered design principles.

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

Figure 5.1 shows an overview of the stages in the development process proposed in our approach. Ubiquitous services are iteratively designed. In each design cycle, the adaptation of the services is put into practice to obtain feedback from the users. The feedback obtained is used to improve the original design. When no further changes are required, the system specification is used to guide the implementation of the final system. This chapter provides detail on the stages involved in the iterative design of the interaction obtrusiveness adaptation of services. Specifically, it introduces the different aspects to be considered from requirements to simulation. Chapter 6 provides detail on the

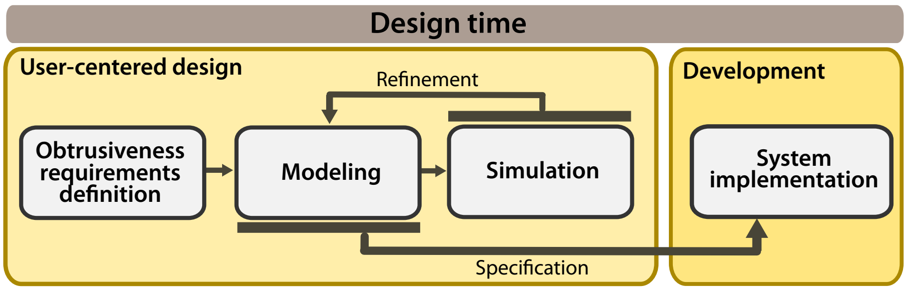


Figure 5.1: The stages proposed in the user-centered development process.

implementation.

The remainder of the chapter is structured in the following manner. Section 5.1 provides an overview of the user-centered design method. Sections 5.2-5.4 introduces the aspects to be considered during each stage of the method. In Section 5.5, a discussion of the usefulness and efficiency of the proposed method is introduced. Finally, Section 5.6 concludes the chapter.

5.1 User-centered design method overview

This section provides a more detailed description of the design method introduced in this work. The user-centered design stage is the initial stage in our approach (see Fig. 5.1). Since we are following a model-driven methodology, the specification obtained at design drives the later stages in the development of the system. Thus, the design becomes central to the development method.

The user-centered design method captures by means of models the concepts that are relevant to achieve considerate services' interaction adaptations. The benefits of using a model-driven paradigm and the steps to be followed in the design of considerate services interactions are introduced below.

5.1.1 Why a modeling approach?

Models are used extensively in design, development, and the sciences. They are powerful tools for representing complex structures and relationships for the purpose of better understanding, discussing, or visualizing them (Cooper et al., 2007). Specifically, the Human-Computer Interaction (HCI) community has considered the use of models to describe interaction for long. In a context where the possible combinations of users, situations and devices are constantly increasing, the implementation of ad-hoc solutions to cover all possible combinations is not feasible (Sottet et al., 2006). Furthermore, a flexible interaction within changing contexts and situations is needed in ubiquitous environments. Thus, the use of models has been identified as a promising approach to handle such increasing complexity (Blumendorf et al., 2010b).

Modeling technologies have been used to formalize distinct aspects of the interaction at different levels of abstraction, and then, to derive the actual code automatically. In this way, the specified system can be automatically generated for different contexts from an abstract description. Currently, the attention has been put on utilizing the models at runtime. The idea stated by Sottet et al. (Sottet et al., 2006) is to keep the models alive at runtime in order to make the design rationale available at runtime. Preserving the models and thus their knowledge at runtime allows reasoning about the designer's decisions.

Because we are designing for different degrees of interaction obtrusiveness according to user's context and we want to adapt it at runtime, it is important to handle this complexity in an understandable and flexible way. Thanks to the use of models, ubiquitous services that are capable of adapting their interaction in terms of obtrusiveness can be defined by working with abstract concepts. Abstraction is one of the fundamental principles of software engineering in order to master complexity (Kramer, 2007). 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 the new interaction components in an automated fashion.

We also leverage these design models at runtime to provide services with the knowledge required for adapting their interaction obtrusiveness dynamically. In this way, the modeling effort made at design time is not only useful for producing the system, but also provides a richer semantic base for self-adaptive behavior during execution.

To summarize, modeling techniques are applied in our approach to obtain the following benefits:

- **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.

- **Reuse of the design knowledge at runtime.** 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. By capturing the knowledge in models, the system can be evolved by simply changing the models at runtime.

Our approach involves to manipulate models in different manners through the different steps. An overview of the steps involved in the design process is provided below.

5.1.2 Steps of the user-centered design process

The design method proposed in this work supports the (1) obtrusiveness requirements definition, (2) modeling, and (3) simulation of considerate service interactions. The method defined involves two development roles: *analysts* and *designers*. Each one makes decisions at different abstraction levels. Figure 5.2 details the tasks involved during each phase of the design method. The steps performed in each phase of the cycle and how the different tasks are involved are detailed below:

1. **Obtrusiveness requirements definition.** *Analysts* are in charge of detecting and defining the obtrusiveness requirements relevant for the final system. In order to gather qualitative data about the users, analysts **make interviews and observations**. From this information, they must **define the personas** (user profiles) that are going to use the system. Personas are used to understand the users and detect their needs and preferences. From the definition of personas, analysts have to **detect the services that our personas require and their obtrusiveness** according to the user's context. Services become the basic unit of work during development. More detail regarding the definition of this aspects is provided in Section 5.2.

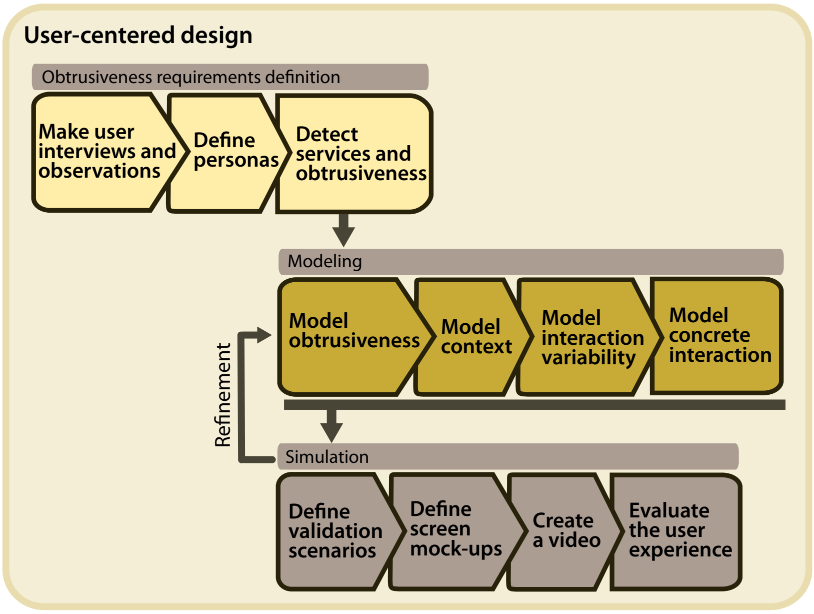

User-centered design
Obtrusiveness requirements definition
Make user interviews and observations
Define personas
Detect services and obtrusiveness
Modeling
Model obtrusiveness
Model context
Model interaction variability
Model concrete interaction
Refinement
Simulation
Define validation scenarios
Define screen mock-ups
Create a video
Evaluate the user experience

Figure 5.2: The different tasks in the design method proposed.

2. **Modeling.** *Interaction designers* are in charge of modeling the attentional demand required for each service according to the personas analysis. This is done by **modeling the obtrusiveness degree** required for a service. In the case that a service could be performed at different obtrusiveness levels, designers must specify the transitions that define how the obtrusiveness of a service can change at runtime. Transitions that change the obtrusiveness level of services are defined with a user situation as a trigger and these user situations are determined based on user's context. Thus, designers have to **model the context** and specify the rules that define each user situation. Then, they have to **model the interaction variability** of the system and select the appropriate interaction resources to interact with the services in each obtrusiveness level supported. Once interaction obtrusiveness adaptation is specified in an abstract manner, *designers* have to **model**

the concrete interaction components that are going to represent the user interface elements available for a specific platform. Composing these concrete components, interaction features are supported for a specific platform. Further detail regarding each one of these steps is provided in Section 5.3.

3. **Simulation.** Designers perform several design iterations to obtain feedback from end-users and improve the obtrusiveness design. Before efforts are put into implementation, they can use the models defined to elaborate a prototype. This prototype can be used to evaluate the adaptation of the system to one or several simultaneous factors. First, designers **define several validation scenarios** according to the user needs and the adaptation that is being designed. According to this scenario, **designers define the screen mock-ups** required in the different contexts and they also **create a video prototype** of the proposed scenario to show the key concepts. Using these fast-prototyping techniques, feedback is gathered from end-users in order to determine whether the proposed obtrusiveness adaptation improves the existing services in terms of **user experience**. Section 5.4 provides the guidelines to perform the simulation.

The design method introduced has been defined to guide the designers in their activities. We have applied model-based technologies to provide support to the different tasks involved in the process. By capturing user needs and obtrusiveness adaptation requirements for the services' interaction in different models, model-based tools can be used to define, validate and guide the development of these considerate adaptive interactions. Once the models have been adjusted to fit with the user needs, a final software solution can be obtained. The derivation of a software solution from the system specification is described in Chapter 6, and detail on the tool support and the validation of the system specification is provided in Appendix A.

The following sections provide further details on each one of the design stages, describing the concepts that are captured to specify the considerate services' interactions. These concepts are later used to obtain an implementation and guide the adaptation of the interaction at runtime.

5.2 Obtrusiveness requirements definition

The first step in the design phase is to understand the users and capture their needs and preferences. This information will determine the obtrusiveness behavior of the different services according to each user needs. The activities involved in this stage are detailed below.

5.2.1 User interviews and observations

Users of a software system should be the main focus of the design effort, thus, it is important to understand their needs and how they are going to use the system. Interviewing and observing users give analysts insights about how the user behaves and thinks about things and the effect that the system may have on the user (Cooper et al., 2007).

Important information to learn from users in order to help in the system specification includes:

- The current behavior of users in their daily life.
- The context of how the different ubiquitous services fits into their lives: what, where, when, why, and how the services will be used.
- Current services that users have and how they use it.
- Goals and motivations for obtrusiveness adaptations or new services.
- User's background in technology and ubiquitous devices.
- Problems and frustrations with current services and their behavior (e.g., intrusion problems).

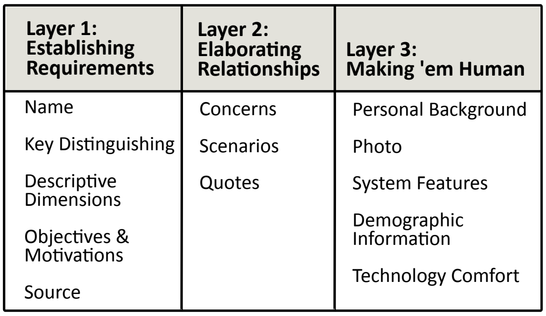
As most people are incapable of accurately assessing their own behavior when they are outside the context of the situations, it is recommended to combine interviewing with observation. This allows designers to clarify questions and ask direct inquiries about situations and behaviors they observe in real time.

This activity falls out of the scope of this thesis. Any user research technique can be used to identify these requirements. Specifically, we have based the analysis on the ethnographic interviewing techniques proposed by Cooper et al. (Cooper et al., 2007). They provide general methods and tips for preparing and conducting ethnographic interviews for a successful design. It is worth noticing that this activity is very important for designers to truly understand the users, their needs, and their motivations, and also to obtain a solid design concept based on qualitative user research.

5.2.2 Persona definition

Having analyzed the users to understand their lives, motivations, and environments, the next step is to describe all the information captured into descriptive models of users. To this end, we make use of **Personas** (also known as User Profiles). A persona is a summary representation of the system's intended users, often described as real people (Brown, 2010). They provide a framework for describing the target audience in a way useful to design and personalize systems.

Personas are used to synthesize the relevant information of the audience to help drive modeling and detect common functionalities between users. From the software engineering side, different mechanisms exist in order to define the relationship between users and their performed activities such as UML Use Case Diagrams (Rumbaugh et al., 1998a), ConcurTaskTrees (Mori et al., 2002), and Business Process Modeling Notation (BPMN) (OMG, 2006a). However, as Cooper states (Cooper et al., 2007), personas are the strongest models that can serve as tools for the interaction designer and they are usually used in the design of user-centered approaches. According to the users, personas give a much more concrete picture of typical users, providing features that directly



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

Figure 5.3: The elements of a persona prioritized into three layers

address specific user needs (Gulliksen et al., 2005). Thus, it is interesting the use of them in this work where we directly address specific user needs.

Personas describe target users of the system, giving a clear picture of how they are likely to use the system, and what they will expect from it (Brown, 2010). They capture relevant information about users that directly impact on the modeling process: user goals, scenarios, tasks, and the like. Although these user profiles are depicted as specific individuals because they function as archetypes, they represent a type of user. Users are grouped into personas, and the personas are analyzed to facilitate service personalization at a person level.

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

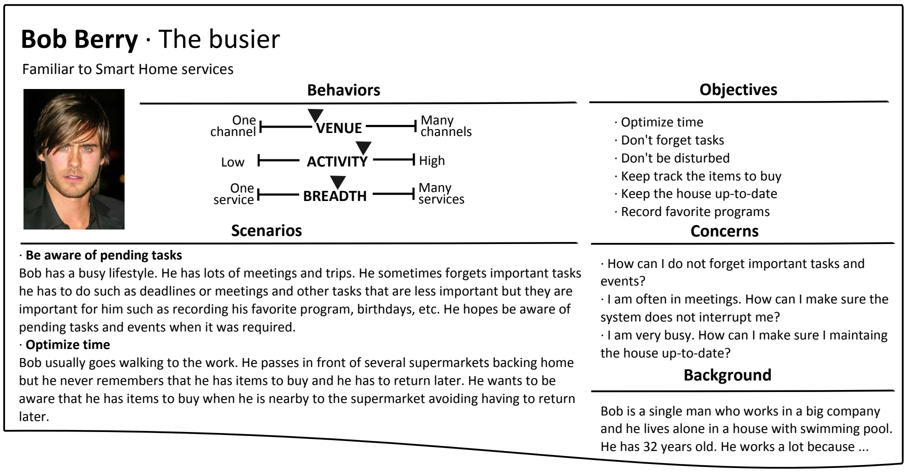
There is no standard format for personas, and different approaches are offered. Regardless of the selected approach, personas should express what users need and what they expect, containing the majority of the user research findings. In this work, we follow the notation defined in (Brown, 2010) to determine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

In this notation, the information is structured following three layers of detail. Figure 5.3 shows the elements of a persona prioritized into these three layers. Layer 1 contains the fundamental elements to establish user requirements. These elements are: the *name* of the persona, some *key features* that distinguish the user group from others, *descriptive dimensions* that are individual scales representing knowledge, tasks, interests and characteristics, the *objectives and motivations* of the persona within the scope of the system and annotations of the data *sources*. These elements can be complemented with information of the other layers such as the *concerns* of the personas that will influence their experience with the system, the *scenarios and circumstances* that set the stage for an interaction between a user and a system, the *personal background*, a *photograph* of the persona, etc.

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

Figure 5.4 shows an example of a persona for a Smart Home system. This persona gives a detailed picture of a typical "busy user" that wants to use Smart Home services to simplify his life and to help him in optimizing his time. This excerpt of a persona provides the basics of user's needs and behaviors. Through careful analysis of this persona, designers can deduce that (1) the user wants Smart Home services for helping him in home tasks not to waste time, (2) he wants to be aware of pending tasks related to home and work, (3) he hopes the system does not disturb him when he is busy, and (4) he prefers as many services as possible.

Information captured in the personas model corresponds to the user requirements or needs structured in goals-scenarios-system features. Designers use this understanding of people to determine *what* services personas require to accomplish their goals and *how* the services


Bob Berry · The busier

Familiar to Smart Home services

Behaviors

One channel
VENUE
Many channels
Low
ACTIVITY
High
One service
BREADTH
Many services

Objectives

· Optimize time
· Don't forget tasks
· Don't be disturbed
· Keep track the items to buy
· Keep the house up-to-date
· Record favorite programs

Scenarios

· **Be aware of pending tasks**
Bob has a busy lifestyle. He has lots of meetings and trips. He sometimes forgets important tasks he has to do such as deadlines or meetings and other tasks that are less important but they are important for him such as recording his favorite program, birthdays, etc. He hopes be aware of pending tasks and events when it was required.

· **Optimize time**
Bob usually goes walking to the work. He passes in front of several supermarkets backing home but he never remembers that he has items to buy and he has to return later. He wants to be aware that he has items to buy when he is nearby to the supermarket avoiding having to return later.

Concerns

· How can I do not forget important tasks and events?
· I am often in meetings. How can I make sure the system does not interrupt me?
· I am very busy. How can I make sure I maintaing the house up-to-date?

Background

Bob is a single man who works in a big company and he lives alone in a house with swimming pool. He has 32 years old. He works a lot because ...

Figure 5.4: Excerpt of a persona

are presented in terms of obtrusiveness. This is done by the designer manually since there is no explicit characteristic of the impact of obtrusiveness on the requirements. Then, this information will be formalized in the models of the next phase by the designer in order to be automatically processed in the development phases.

5.2.3 Services and obtrusiveness definition

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

For example, the services detected from the synthesis of the persona of Figure 5.4 are: a *shopping list* to keep track the items to buy, an *agenda* that notifies him important tasks, a *video recorder* that records his favorite programs, and a *supermarket notification* to remember him that he has items to buy.

Service	**Attentional Demand**	**Context to Consider**
Shopping List	low attention, high attention	user activity (shopping, eating, cooking)
Agenda	slightly attention, high attention	message priority, deadline of task, user status (alone, with company), user activity (working, in a meeting, etc.)
Video Recorder	low attention, slightly attention	program priority, user activity (watching the TV, out of home)
Supermarket Notification	slightly attention, high attention	location of user, number of items to buy, user activity (driving, walking, running)

Table 5.1: Service analysis for the Bob Persona

For these services detected, the degree of attentional demand required according to the user context is: *low attention* for managing the shopping list, although the user can add items manually, *slightly or low attention* for the video recorder service depending on the program to record and the user situation whether he is watching the TV or he is not at home, and *high or slightly attention* for the supermarket notification. The agenda to notify important tasks could require *slightly* or *high attention* depending on the priority of the task for the user and his user situation (working, in free time, in a meeting, etc.). The priority of the task can be set up by the user in their preferences. Table 5.1 shows a summarized view of the service analysis results for the Bob persona. The table shows the services needed for the persona, the attentional demand and the contextual aspects that can affect the obtrusiveness adaptation.

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

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

5.3 Modeling

Once the obtrusiveness requirements are captured, the different models that characterize the interaction obtrusiveness adaptation are defined. First, the attentional demand required for each service is defined in terms of obtrusiveness according to the personas analysis. Also, the context is formalized by defining the properties that can affect the user's situation. Once the possible obtrusiveness levels for each service are specified, the appropriate interaction resources can be selected from the ones available. These abstract models are complemented with others that provide a more concrete representation of the service components such as the concrete interaction components that are going to represent the user interface elements available and optionally the architecture of the components that form the system (this model is only used for implementation purposes and introduced in Chapter 6). The *interaction designer* is the role in charge of this phase (henceforth, we well refer to it as designers).

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

5.3.1 Obtrusiveness modeling

We make use of the implicit interaction framework presented in (Ju and Leifer, 2008) to determine the **obtrusiveness level** for each interaction in the system. This framework helps designers to create interactions that are more socially appropriate, focusing on the *manner* in which the devices interact with the user. Figure 5.5 shows the two dimensions to characterize implicit interactions: *attentional demand* and *initiative*. Attentional demand is the degree of cognitive and perceptual load imposed on user by the interactive system. According to this factor, *foreground* interactions require a greater degree of focus, concentration and consciousness, (the user is fully conscious of the interaction) while *background* interactions are peripheral and elude the user's attention (the user is unaware of the interaction with the system). Interactions that are initiated and driven by the user explicitly are called *reactive* interactions, while interactions initiated by the system based on inferred desire or demand are *proactive* interactions.

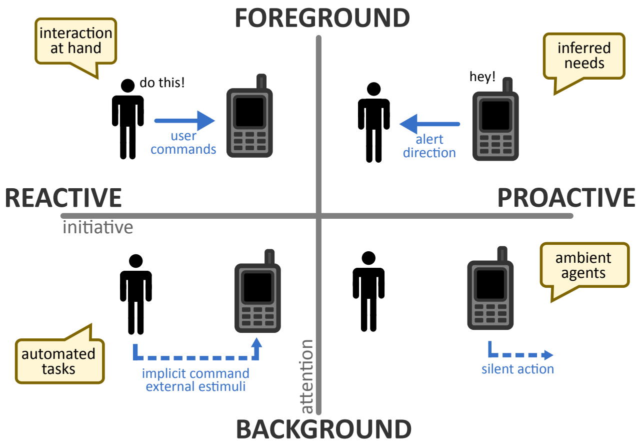


Figure 5.5: Conceptual framework used for the definition of implicit interactions.

Other frameworks and taxonomies on automation levels were studied. Sheridan and Verplanck (Sheridan and Verplank, 1978) developed a taxonomy to classify the level of automation. The taxonomy incorporated issues of feedback (what the human should be told by the system), as well as relative sharing of functions determining options, selecting options and implementing. Alternative forms of this taxonomy were discussed in the literature (Endsley and Kaber, 1999; Wei et al., 1998). Endsley and Kaber (Endsley and Kaber, 1999) demonstrated the level of automation effects on performance, situation awareness, and workload in a dynamic control task. Wei et al. (Wei et al., 1998) suggested a model for the appropriate degree of automation of different tasks based on a tasks effect on system performance and its demand on the operator relative to other tasks. However, all of these scales of degrees of automation focus on the traditional explicit human-machine interaction and do not address the implicit interaction issue. Other frameworks exist for the definition of implicit interactions (Horvitz et al., 2003; Buxton, 1995). However, we found it very useful to consider initiative and attention as independent concepts. In the case of mobile interaction adaptation, automation (initiative) and user awareness (attention) are factors that usually vary independently from service to service (e.g., an automated task can require the user to be aware of it or not depending on different contextual factors such as the user workload or the user location).

According to our proposal, once the services are defined for each persona, designers must indicate the possible obtrusiveness levels for each service according to user needs and the context of use. The conceptual framework used in this work defines two axes to represent obtrusiveness. For each persona, we take this space and make different divisions of each axis. This constitutes the **unobtrusive adaptation space**. Designers can divide this space into many disjoint fragments as they need to provide specific semantics to each fragment. 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 our approach, we use these divisions to drive the selection of the interaction mechanisms that are better suited for each situation.

Figure 5.6 shows the linkage between different services for the Bob persona (see Table 5.1) and the unobtrusive adaptation space for the interactions that support the services. The initiative axis in this case

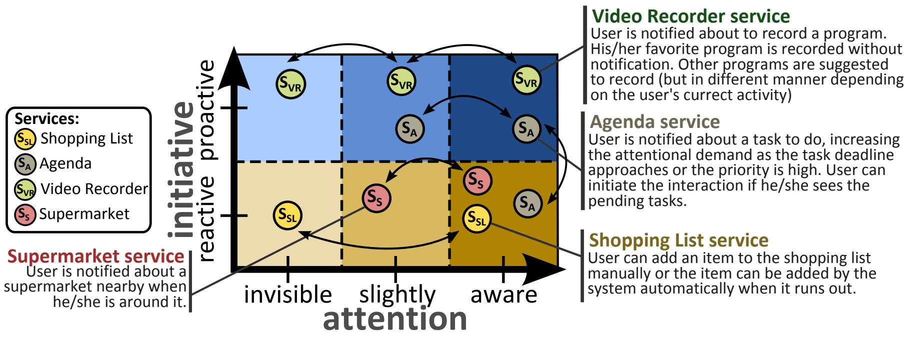

Video Recorder service
User is notified about to record a program. His/her favorite program is recorded without notification. Other programs are suggested to record (but in different manner depending on the user's currect activity)
Agenda service
User is notified about a task to do, increasing the attentional demand as the task deadline approaches or the priority is high. User can initiate the interaction if he/she sees the pending tasks.
Shopping List service
User can add an item to the shopping list manually or the item can be added by the system automatically when it runs out.
Supermarket service
User is notified about a supermarket nearby when he/she is around it.
Services:
S_{SL} Shopping List
S_A Agenda
S_{VR} Video Recorder
S_S Supermarket
initiative
reactive
proactive
invisible
slightly
aware
attention

Figure 5.6: Services at different obtrusiveness level according to the Bob persona

is divided in two parts: *reactive* and *proactive*. The attention axis is divided in three segments which are associated with the following values: *invisible* (there is no way for the user to perceive the interaction), *slightly-appreciable* (usually the user would not perceive it unless he/she makes some effort), and *user-awareness* (the user becomes aware of the interaction even if he/she is performing other tasks). Designers can divide each axis in as many parts as they require for describing the obtrusiveness level of the services.

In this particular example, the services needed for Bob are located in the unobtrusive adaptation space according to his needs and contextual information detected in the previous stage. For example, the service to record a TV program is offered at different obtrusiveness level depending on different context conditions. The program is recorded automatically in an invisible manner if it is the favorite program of the user and he is not watching it. For other programs, according to the user preferences, a suggestion is provided to the user about the possibility to record it. This suggestion interrupts the user in a different manner depending on the activity he is engaged in. For example, if the user is attending a meeting or driving, suggestion will be provided in a subtle manner (*slightly-appreciable* level of attention) in order not to disturb him. Another services, such as the agenda, can increase the attentional demand required as the deadline of a notification approaches if the user is not still aware of it.

These divisions allow designers to classify the different services according to the interaction obtrusiveness required at each moment, taking into account implicit and explicit interactions. In this way, designers can later choose the interaction modalities that best fit these obtrusiveness requirements captured. More detail on the interaction modalities chosen is provided in the next section.

Defining transitions

During runtime, interaction with a service is offered in only one of the obtrusiveness levels that are determined by the partitions of the unobtrusive adaptation space. Nevertheless, this obtrusiveness level can be modified in response to changes in the user situation. Designers can define *transitions* (see Figure 5.6) that link user situations with changes in the obtrusiveness level. Each transition is composed by a *user situation* and an *action*. When a user situation is fulfilled, the obtrusiveness level is modified by changing the attention level, the initiative level, or both, as defined by the *action*.

In this work, *user situations* are expressed as logic rules that check for values in the ontology-based context model. Thus, in order to allow the definition of transitions, the corresponding rules must be defined and the context information that use the rules must be included in the context model. More detail regarding the definition of the rules for defining the user situation is provided on the next subsection (5.3.2 Context modeling).

Adaptation actions produce an impact on the obtrusiveness level of a service. When a user situation of a given transition is fulfilled, the corresponding action is performed. An action is defined as a tuple *(obtrusivenessAxis, destinationLevel)*, where the *obtrusivenessAxis* is the name of the axis (e.g., Attention or Initiative) and *destinationLevel* is a particular value for an axis.

The *destinationLevel* can be expressed in either a relative or absolute manner. A *relative* destination is specified as an increment or decrement of the obtrusiveness level for one or both axes (e.g., an increment of two levels for attention and a decrement of one level for initiative). An *absolute* destination represents the particular value for one or both axes. For example, the user situation *withCompany* is associated to the transition T as follows (an absolute *destinationLevel* is used):

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

This means that when the system senses the user is with company (according to the user situation), it must adapt the interaction for the service associated to this transition to adjust the obtrusiveness level to *(slightly-appreciable, proactive)* level.

The use of relative or absolute actions depends on the specific semantics that designers are using. Nevertheless, the relative actions allow to specify changes in the obtrusiveness level that are sensitive to the current obtrusiveness level. For example, in the agenda service, we can define a *relative* action that produces an increment in the *attention* axis each time a deadline approaches and the user is not still aware of it.

In the case that multiple user situations are fulfilled at the same time, contradictory actions may be triggered, i.e., a movement in the unobtrusive adaptation space performed in different (possibly opposite) directions. To resolve this conflict, actions are aggregated before they are applied. This process involves the following steps:

1. *Absolute actions are expressed as relative actions.* This is done by calculating the increment that is required to reach the desired destination from the current obtrusiveness level. For example, if the attention axis is divided in three parts and the current task is performed at the *invisible* level, an action that requires attention to be at the *aware* level is represented as an increment of 2 units for this axis.

2. *Actions are aggregated.* The median is calculated for all the relative increments of the different actions for each axis to obtain

an average increment. The rationale behind this is to obtain the average movement in the unobtrusive adaptation space. We use the median instead of the mean in order to avoid extreme results to affect the changes in the obtrusiveness level.

3. *The resulting action is applied.* The current initiative and attention levels are modified according to the increment obtained by aggregating the different actions. In this way, only a single obtrusiveness change is performed.

5.3.2 Context modeling

Mobile and ubiquitous devices accompany users throughout all of the day, operating within a context of significant constraints and environmental distractions. They need "awareness" of several contextual factors including social, psychological, physical, and the like (Tamminen et al., 2004). Different types of context can be considered for mobile and ubiquitous computing (Maiden, 2009):

- *Computing context* is everything related to computational resources, such as available networks, network bandwidth, communication costs, and nearby computational resources such as printers.
- *User context* is information about the user interacting with the device, such as a profile (e.g., age), location (e.g., geographic position), proximity (e.g., distance to another person), preferences, skills, etc.
- *Physical context* involves factors in the environment of the device with which the user interacts, such as temperatures, noise levels, and speed and lighting levels.
- *Time context* involves information such as absolute time, date, and day of the week.

According to the previous classification of contextual information, the adaptation of the interaction obtrusiveness deals directly with information related to the *user*, *physical* and *time context*. For example, the presence of a close friend in the nearby area (user context) can be notified to the user at a *foreground* level of attention compared to the presence of other people with a further distance in their social network which can be queried on demand by the user (i.e. *reactive* behavior is used). *Computing context* could also be considered, adapting interaction according to the different devices that the user possesses, but we do not consider this adaptation in the present work.

For context modeling, we use an *ontology-based context model* that leverages Semantic Web technology and OWL (Web Ontology Language) [1]. OWL is an ontology markup language that enables context sharing and reasoning. In the Artificial Intelligence literature, an ontology is a formal, explicit description of concepts in a particular domain of discourse. It provides a vocabulary for representing domain knowledge and for describing specific situations in a domain. An ontology-based approach for context modeling lets us to describe contexts semantically and share common understanding of the structure of contexts among users, devices, and services. The main benefit of this model is that it enables a formal analysis of the domain knowledge, such as performing context reasoning using first order logic.

The ontology used in this work is described in OWL as a collection of RDF[2] triples, in which each statement is in the form of *(subject, predicate, object)*. The subject and object are the ontology objects or individuals and the predicate is a property relation defined by the ontology. For instance, *(Bob, usersInLocation, bedroom)* means that Bob is located in the bedroom. Figure 5.7 shows the class diagram of our used ontology. Information regarding the user environment, the services of the system, the users, the temporal aspects and the events that happen in the system are represented in this ontology. For more information about the structure and population of this ontology see (Serral, 2011).

It is important to note that we have used this ontology because it covers the core context needed for interaction adaptation purposes. However, other context information specific for a system may be needed.

[1] http://www.w3.org/standards/techs/owl

[2] http://www.w3.org/RDF/

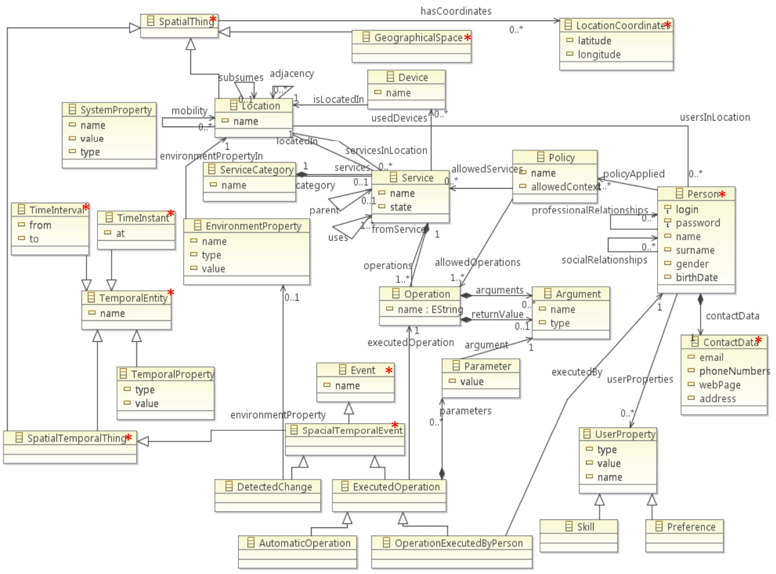


Figure 5.7: OWL Ontology classes (Serral, 2011).

In this case, the ontology can be easily extended with new classes to cover this information. In particular, we have extended the ontology by adding the *userSituation* class to properly describe the current situation of the user, and the relationship *currentSituation* that link the *Person* class with the *userSituation* class. The information of this class is inferred by means of rules as explained in the next subsection.

Figure 5.8 shows an example of the context model using a graphical tree representation in OWL format. In this figure, some of the classes and properties of the used context ontology are shown as well as some individuals created as examples.

Defining user situations

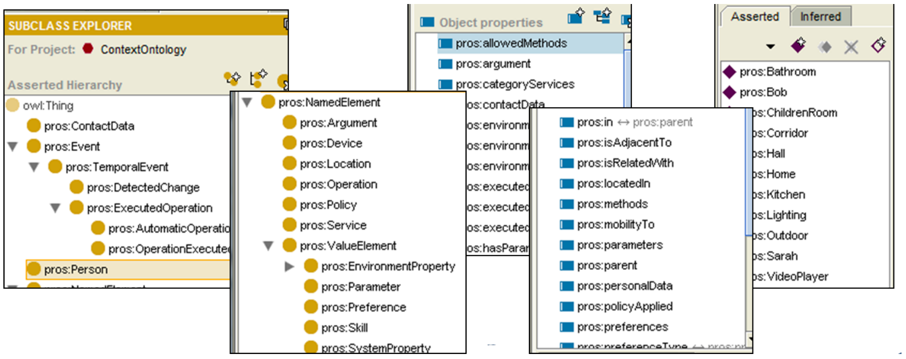


Figure 5.8: An example of the OWL ontology context model.

Given the above ontology, the context processing mechanism that is in charge of keeping contextual information consistent with the real world and inferring the current user situation is based on logic rules. These rules aggregate and filter low level information and generate meaningful events or user situations. For example, Listing 5.1 shows a rule to infer when the user is with company:

Listing 5.1: Example of the *withCompany* rule.

```
[withCompany: (?user rdf:type pros:Person)
      (?user pros:usersInLocation ?location)
      (?person1 rdf:type pros:Person)
      (?user pros:knows ?person1)
      (?person1 pros:usersInLocation ?location)
      (?user pros:socialRelationships ?person1)
      ->
      (?user pros:currentSituation pros:withCompany)]
```

By aggregating and filtering context events we can obtain user situations that are relevant for the interaction obtrusiveness adaptation. For example, a change in the user's location can trigger the change of the information of another entity (*pros:currentSituation*).

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 *Situation Specification Interface*. This is further explained in Chapter 7.

5.3.3 Interaction variability modeling

Depending on the obtrusiveness degree in which a service is performed, interaction will be offered in a different manner. The obtrusiveness level determines the type of interaction offered to the user. Thus, the appropriate use of a modality or modality combinations can play a crucial role, attenuating the required attention (de Sá et al., 2010). In this way, we aim to adapt interaction in terms of obtrusiveness to the user's current environmentally attentional resources by choosing the appropriate interaction techniques.

Devices can interact with users through different modalities. Modality taxonomies have been proposed to serve as theoretical foundations for understanding and describing modalities (Bernsen, 1994; Bachvarova et al., 2007; Chittaro, 2010). A modality is a way of exchanging information between humans and machines in some medium (Bernsen, 1994). According to Bachvarova (Bachvarova et al., 2007), at the perception level we can distinguish between *visual*, *auditory* and *haptic* modalities. Visual are the modalities that are perceived through the visual sensory channel, for example written text or images. Auditory modalities are perceived through the auditory sensory channel, for example speech or music. Haptic modalities are related to the sensory system of touch.

However, there is not a universal interaction technique that is well suited for any situation. Studies have been made to evaluate the effects on the cognitive load of the different modalities and to detect the appropriate combinations of modalities (Mayer and Moreno, 2003; Cao et al., 2009; Haapalainen et al., 2010). Results shown that the selection of an appropriate interaction modality depends on (1) the **user situation** and (2) **their effects on cognitive load**.

Table 5.2 presents a summarized information of the output modalities considered in this work based on the existing multimodal design taxonomies and frameworks (Bernsen, 1994; Bachvarova et al., 2007; Obrenovic et al., 2007; Lemmelä et al., 2008). This table shows the

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
Manifesta-tions	Graphical icon, text, image, graph, map, lights	Beep, synthetic speech, music, acoustic alarm	Vibration, force feedback, temperature

Table 5.2: Summarized information of output modalities

strengths, limitations and properties of the different modalities. 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 helps to model the interaction variability in terms of obtrusiveness.

Feature Modeling

In order to define the interaction variability of ubiquitous and mobile devices according to the obtrusiveness levels, we make use of **Feature Models** (Czarnecki et al., 2004). Specifically, a feature model is used in our approach to represent interaction modalities and modality combination as well as describe their variability. Feature models enable us to specify not only current interaction features of a system but also potential features since they may be activated in the future. We argue that in response to changes in the obtrusiveness level, the system itself can query this feature model in order to determine the necessary modifications to its interaction components. For example, a notification service can trigger the activation of both visual and auditory features when it requires more attention of the user due to the priority of the message or the user situation (e.g., user alone).

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 (Benavides et al., 2005). Feature modeling is widely used to describe a system configuration and its variants in terms of features (coarse-grained system functionality). In our work, a feature is a distinctive user-visible aspect or characteristic of the interaction. In these models, features are hierarchically linked in a tree-like structure through variability relationships such as *optional*, *mandatory*, *single-choice*, or *multiple-choice*, and are optionally connected by cross-tree constraints such as *requires* or *excludes*.

Besides describing the relevant aspects of the system, feature models have proven to be effective in hiding much of the complexity in the definition of the adaptation space (Cetina et al., 2009). We make use of feature models to reflect the terms in which the interaction is perceived in an abstract manner and the constraints that exist for their selection.

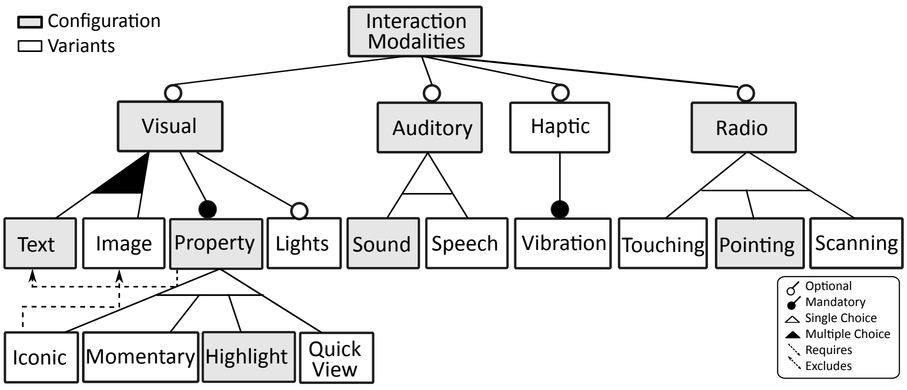


Figure 5.9: Feature model of output interaction modalities.

Using feature models, we 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 situations. 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 feature model based on the studies previously mentioned. Figure 5.9 shows the defined feature model to represent visual, auditory, and haptic modalities and the constraints for their selection. It also includes radio modalities that correspond to the kind of interactions between the users and the physical elements. For example, users can access the services that are associated with an element either by *pointing* to the element, *touching* it, or *scanning* nearby elements with their mobile device. These are only some examples of the interaction techniques that have been defined for the interaction between users and their surroundings in the literature (Broll et al., 2008; Rukzio et al., 2006).

The grey features of Figure 5.9 represent the *Interaction Configu-*

ration (IC) for a service in a particular obtrusiveness level, while the white features represent potential variants since they may be activated in the future if another obtrusiveness level is required. For example, the user can be in a meeting and interaction with a service should be adapted to this situation, offering a more subtle interaction with the system (e.g., deactivating the sound modality and activating the vibration modality). The IC of the feature model of Figure 5.9 is defined as follows:

$$IC_{Figure 5.9} = \{InteractionModalities, Auditory, Sound,$$

$$Visual, Property, Highlight, Text, Radio, Pointing\}$$

Each IC of the feature model is defined by the set of feature states. The feasible feature states are: *active* and *inactive*. It is the task of designers to define the possible *interaction configurations* in which a feature model can evolve. These configurations are validated using the *FeAture Model Analyzer* Framework[3] (FAMA) in order to ensure a consistent interaction adaptation. More detail about this validation is provided in Appendix A.

The definitions that are contained in the feature model are by no means considered universal. The feature model is intended to capture the perspective that designers have about interaction. It can be used to model another interaction techniques and constraints adapted to each particular project requirements. In the example, we have considered that an interaction element can be either visual or auditory, which is obviously a simplification since many common widgets normally combine these aspects (e.g., to offer feedback to the user).

Mapping to obtrusiveness

Once the unobtrusive adaptation space and the interaction variability are modeled, we relate these models to match the appropriate interaction configuration to each service in an obtrusiveness level. Specifically,

[3]http://www.isa.us.es/fama/: FAMA framework

this relationship is represented by means of a dependency relationship between the metaelements of the different metamodels, i.e, an *obtrusiveness level* element has associated a set of features (*configuration* element) of the feature model.

This process to select the modalities that best convey the interaction context is called *Modality Allocation* (Karagiannidis et al., 1997). This selection is based on the studies of the effects on cognitive load of each modality and modality combinations previously mentioned. For example, an image uses less mental workload than a text, and visual-auditory combinations impose less cognitive load than visual-visual combinations (Cao et al., 2009). The auditory modalities are useful for attention alerting (Reeves et al., 2004), and the vibration or lighting feedback do not interrupt other activities (Savio and Braiterman, 2007).

Each obtrusiveness level will have an IC associated to it. This will determine when a feature must be activated/deactivated (depending on the attention required). For example the IC of the feature model of Fig. 5.9 is assigned to the *(aware, proactive)* obtrusiveness level as follows:

$$Obtrusiveness_{(aware,proactive)} = CC_{Figure\ 5.9}$$

It is worth noticing that for the *invisible* attentional demand we do not assign an explicit configuration since these interactions occur without the explicit behest or awareness of the user. These interactions are more presumptuous because user does not have the opportunity to oversee and possibly cancel the action of the service.

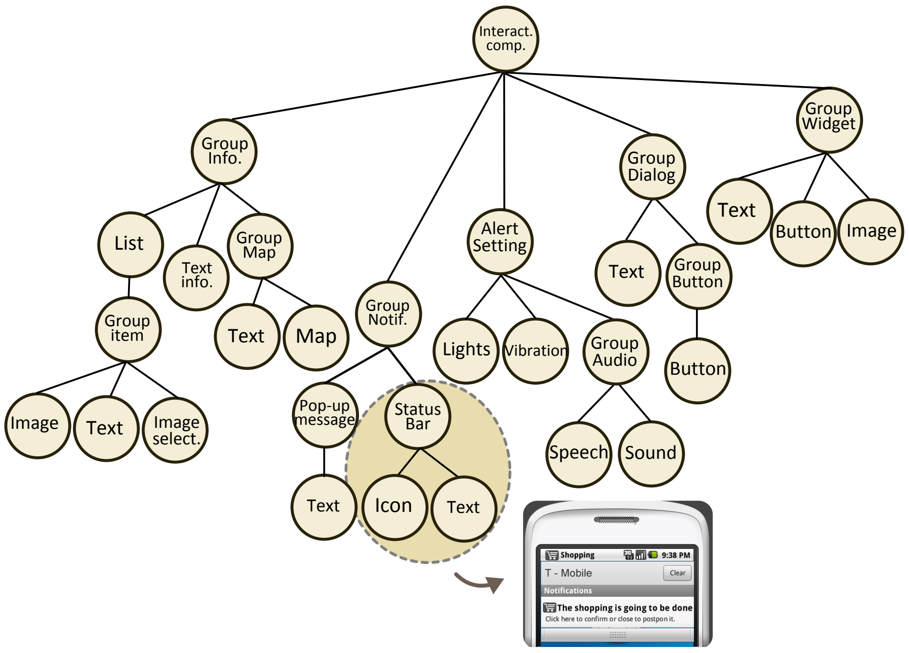
5.3.4 Concrete interaction modeling

Since features represent coarse grained functionality, there is a need to detect which concrete interaction components are represented by each feature. This concretizes the abstract interaction features to address interaction adaptation at a concrete level of abstraction. In this way, we can adapt at abstract level and determine the impact of the change in the concrete components.

For the purpose of this work, we assume a concrete interaction model that is organized in a *tree structure* allowing a flexible composition of the interaction elements. In this structure, components can be contained in other components following a hierarchical representation that allows an easier definition of UIs as seen in iOS or Android UIs. This node-based user interface provides an easier node substitution (to adapt UIs at run-time), since components can be dynamically activated and deactivated. Component-based architectures have been identified for their multiple adaptation possibilities such as addition of new components or interface reconfiguration (Grundy and Hosking, 2000). Moreover, this interface structure can be used for any platform that has a component-based user interface such as Android, iOS or any user interface following the composite pattern.

In our work, the nodes represent concrete interaction objects that perform a specific functionality. They are any UI components that the user can perceive such as graphical objects, text, image viewers, UI controls, video viewers, widgets, and concrete resources available in mobile devices to provide feedback such as the vibration, speech synthetization, sound, etc.

Different representations exist in the HCI community to define the concrete user interface such as UML class diagrams (da Silva and Paton, 2003), the XML User Interface Language (XUL) (Butter et al., 2007), or Concrete Interaction Objects (CIOs) (Limbourg et al., 2004). These descriptions also follow a hierarchical representation of containers and components of the interface. Representing the interfaces using class diagrams can not give an intuitive visualization about the organization of the associated interface. XUL and CIOs are based on the XML markup language to specify the structure of the user interfaces supporting the modalities graphical and auditory. Conversely, our node-based user interface provides a very powerful, flexible, and intuitive way of working with advanced interaction events and an easier support for animation, multi-touch interaction and visual effects. Also, Android introduced *application fragments* in Android 3.0 in order to help applications adjust their interfaces and reuse different parts of the applications' user interface. This was introduced due to the need to support more dynamic



The shopping is going to be done
Click here to confirm or close to postpon it.

Figure 5.10: Concrete Interaction Components model.

and flexible UI designs when considering different conditions such as large screens (tablets, TVs) or new interaction mechanisms. This user interface composition is similar as the one we propose in this work.

An example of the concrete interaction components model is shown in Figure 5.10. Users can interact with the services in several ways depending on the obtrusiveness level selected for the service. In this model, interaction can be offered (from left to right) using a list of items or an interface showing a location map. A notification (*Group Notif.* node) can be offered by a momentary pop-up message (*pop-up* node) or by means of a notification window for persistent reminders that come from the background and request the user's response (*Status bar* node), by an alert dialog that requires confirmation from the user (*Group Dialog* node), or by an app widget to show information in a subtle manner (*group widget* node). In conjunction to these modes, flashing lights (*lights* node), vibration feedback (*vibration* node) and an alert sound (*group audio* node) can be used. All of these components are represented in the tree structure of the figure.

Mapping to interaction features

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

Figure 5.11 shows an example of the mapping between the interaction features and the concrete user interface components. For example, the *iconic* modality is supported by the *Group Notif.* and *Status Bar* nodes. Also, *Speech* modality is supported by the *Alert Setting*, *Group Audio* and *Speech* nodes (see Fig. 5.11 (a)).

To illustrate the role that play the mapping in the adaptation process, Figure 5.11 shows two interaction configurations of the feature model. Part (a) of Fig. 5.11 shows an *(aware, proactive)* interaction configuration for the recording service, while part (b) of Fig. 5.11 shows a (*slightly, reactive*) configuration for the same service. Comparing both configurations, a status bar is used for notifying the user about to record a program in conjunction with speech interaction (a) because the user is totally aware of the interaction. Then, the same notification is presented by means of a widget in the home screen and the speech interaction is changed by lights (b) for a more subtle interaction.

Specifically, we use a *weaving model* (Fabro et al., 2006) to perform this mapping. This weaving approach enables us for scoping and configuring the concrete interaction components model from a set of given features. Weaving models are used to define and to capture relation-

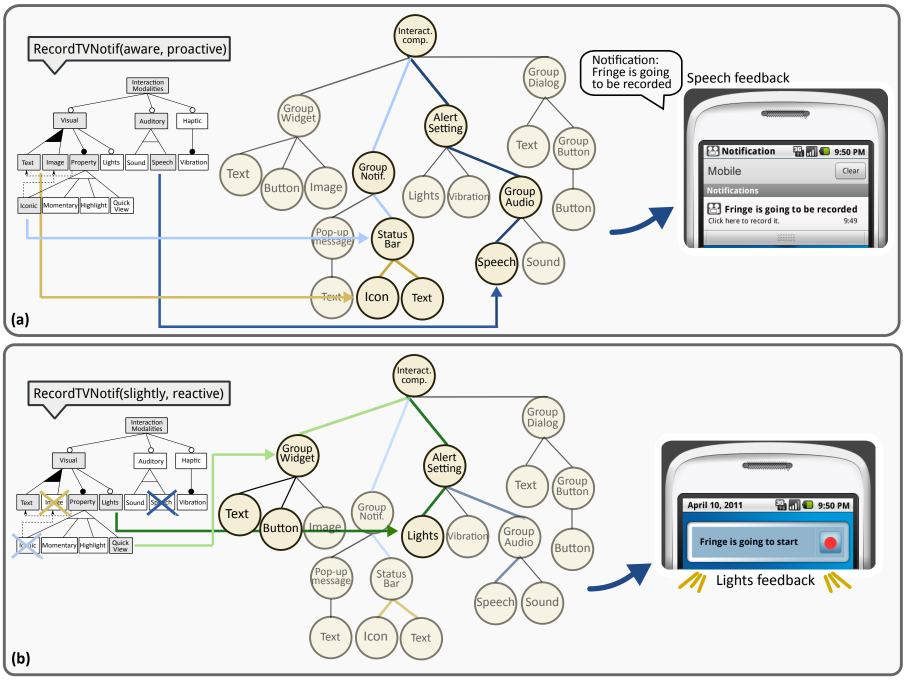


Figure 5.11: Relationship between features and interaction components.

ships between model elements. Relationships between model elements are present in many different application scenarios, such as specification of transformations, traceability, or model alignment. We use the weaving models to define the relationships between features and interaction components of the concrete interaction components model.

Consider a weaving model (Wm) between a Feature model (Fm) and a Concrete Interaction Components model (Im), denoted by the triple $< Wm, Fm, Im >$. Wm contains a set of elements that link a set of elements of Fm with a set of elements of Im. Specifically, the weaving model states that each interaction feature in the feature model has a one-to-many relationship to elements in the concrete interaction components model.

Thanks to the introduction of an abstract description of interaction and the mapping to the concrete interaction components, designers can deal with high level concepts closer to the requirements and they can determine directly the impact on the concrete interaction components that support those requirements.

5.4 Simulation

The previous sections introduced a design method for the definition of adaptive service interactions in terms of obtrusiveness. However, when an adaptive system is designed, there is no guarantee that the resulting adaptation could met the user expectations. The User-Centered Design (UCD) cycle suggests to perform simulations of a scenario before it is finally implemented. Simulations are a kind of drama where devices are props, environments are stages, users are actors, and user experiences have internal narratives (Kuniavsky, 2010).

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

5.4.1 Requirements for the evaluation

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

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

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

Evaluation from the user experience. Interaction design is a key factor that determines the user experience in the interaction between users and services (Kuniavsky, 2010). This aspect is affected by different factors such as effectiveness (how good is the adaptation?), efficiency (how fast is it?) and emotional satisfaction (how good does it feel?). When evaluating interaction obtrusiveness adaptation, these perspectives should be considered: user emotional perception of the adaptability and the productivity increase for the system.

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

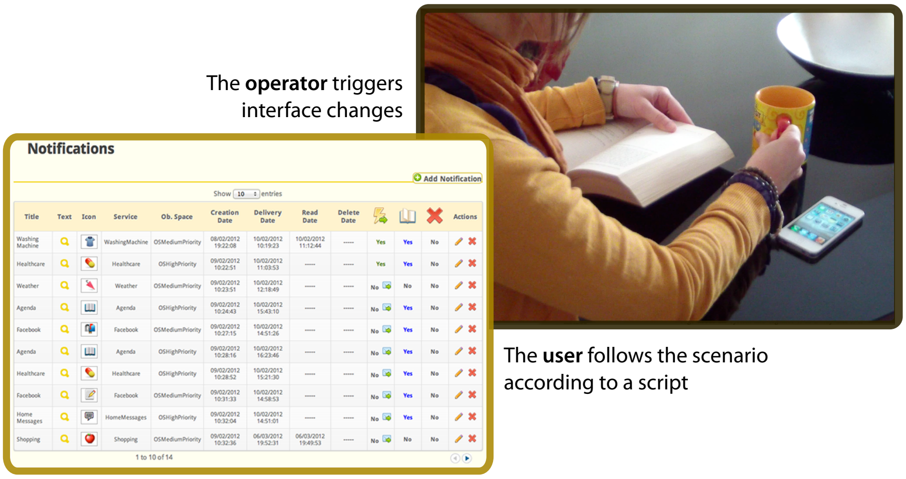
5.4.2 Fast-prototyping for interaction obtrusiveness adaptation

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

A mock-up is designed for each user interface offered in the different contexts. These mock-ups provide the user with the expected interaction given a set of context conditions and user needs. Since the users have to follow a script that conforms a specific role, it is easy to anticipate the results that can be obtained. In addition, the availability of powerful utilities to develop graphical user interfaces make this approach **quick to apply**.

The adaptation of interaction for each service is simulated using Wizard of Oz techniques (Dahlbäck et al., 1993). An operator provides the current user interface according to the user situation using another device (see Fig. 5.12). The operator is in charge of providing the properly user interface depending on the context of the user (e.g., providing a service in the completely aware obtrusiveness level when s/he is having breakfast). In this way, the user is immersed in an environment that behaves like a working system with context-aware capabilities, but it is much easier to produce.

In order to obtain fast-prototypes for interaction obtrusiveness adaptation of ubiquitous services, we have followed the steps described below (see Figure 5.13).


The **operator** triggers interface changes
The **user** follows the scenario according to a script
Add Notification

Notifications

Show 10 entries

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

Figure 5.12: Fast-prototyping evaluation applying Wizard of Oz techniques and mock-ups.

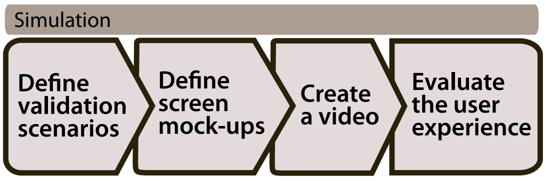


Figure 5.13: The different tasks in the simulation stage.

1. **Define validation scenarios where considerate interaction adaptation is reflected.** Specific scenarios (i.e., mini scripts that describe different situations) are defined to illustrate the way services' interactions are adapted in terms of obtrusiveness. In each scenario, one or several services are presented at different obtrusiveness levels simulating key adaptations in the service interaction behavior. For example, we can define a video recorder service presented first in a *reactive-invisible* manner because it begins to record automatically in reaction to the user leave, and then the same service *proactively* notifies the user about to record

the program in a *subtle* manner.

The resulting description is like a use case or user story in software development, but more detailed and focused on the value of the adaptation experience to people.

2. **Define the screen mock-ups for each scenario.** Depending on the obtrusiveness levels considered in the scenarios, the interaction is offered in a different manner according to the user situation (e.g., as taskbar notification). Thus, the different screens needed for the scenarios must be defined and links between screens must be provided in a decoupled manner. The screen to be shown depends on the current context of the user and his/her attentional resources (i.e., user situation).

3. **Create a video prototype of the proposed adaptations.** A conceptual video prototype is created to show the key concepts of the interaction obtrusiveness adaptation. A video prototype makes users to be familiar with the adaptations and provides a quick exploration of the user experience by using our system. Designers can create a rapid video prototype in hours or few days using consumer-grade video hardware.

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

For the development of mock-ups two opposed requirements are faced. We want mock-ups to be realistic but we also want them to be very easy to develop. To face both requirements, we considered the use of a set of Android and iOS utilities for the development of both Android and iOS GUI applications. There are different WYSIWYG user interface tools to visually design the way the app looks instead of writing code. Some of these tools are the *App Inventor* for Android[4], the *DroidDraw UI Designer*[5], or the *Interface Builder*[6] for iOS. Using these tools, we can quickly design UI layouts and the screen elements they contain, with a series of nested elements. In particular, Android has an XML-based layout file for each user interface. So, user interfaces can be easily defined by means of this layout file.

Specifically, we have developed mock-ups for Android and iOS using the ADT Plugin for Eclipse and DroidDraw UI Designer (for Android) and the Interface Builder (for iOS). On the one hand, the ADT Plugin offers a layout preview for the XML file. Also, we used DroidDraw UI Designer to create the XML file since it is a graphical user interface builder for the Android platform which generates the XML files from UI designs. In particular, we used both utilities to achieve a look-and-feel for Android. On the other hand, Interface Builder provides palettes of user interface objects (e.g., text fields, data tables, sliders, etc.) to develop user interfaces by simply dragging the interface objects from the palette onto a window or menu.

Listing 5.2 shows an excerpt of a XML prototype developed for Android.

Listing 5.2: Excerpt from the Android prototype.

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

<TextView
  android:id="@+id/SupermarketName"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Mercadona"
```

[4]http://appinventor.mit.edu
[5]http://www.droiddraw.org
[6]https://developer.apple.com/technologies/tools/

```
      android:textStyle="bold"
      android:layout_marginLeft="10dip"
      android:layout_marginTop="5dip">
    </TextView>
    <TextView
      android:id="@+id/SupermarketAddress"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Calle del Gorgos, S/N, 46021 Valencia"
      android:layout_marginLeft="10dip">
    </TextView>
    <ImageView
      android:id="@+id/Map"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:layout_marginTop="15dip"
      android:scaleType="center"
      android:src="@drawable/mapa2">
    </ImageView>
    </LinearLayout>
```

The code illustrated above is a layout file that represents one screen mock-up. Each layout file must contain exactly one root element. Once the root element is defined, additional layout objects or widgets can be added as child elements to gradually build a *View* hierarchy that defines the layout. Figure 5.14 shows some mock-ups of the Android prototype. The rendering of the XML layout file showed above is shown in the top right of the figure.

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

5.4.3 Models refinement

In order to obtain valuable feedback from users and evaluate their user satisfaction with the design, a questionnaire is used. This questionnaire uses questions from the IBM Post-Study questionnaire (Lewis, 1995) in

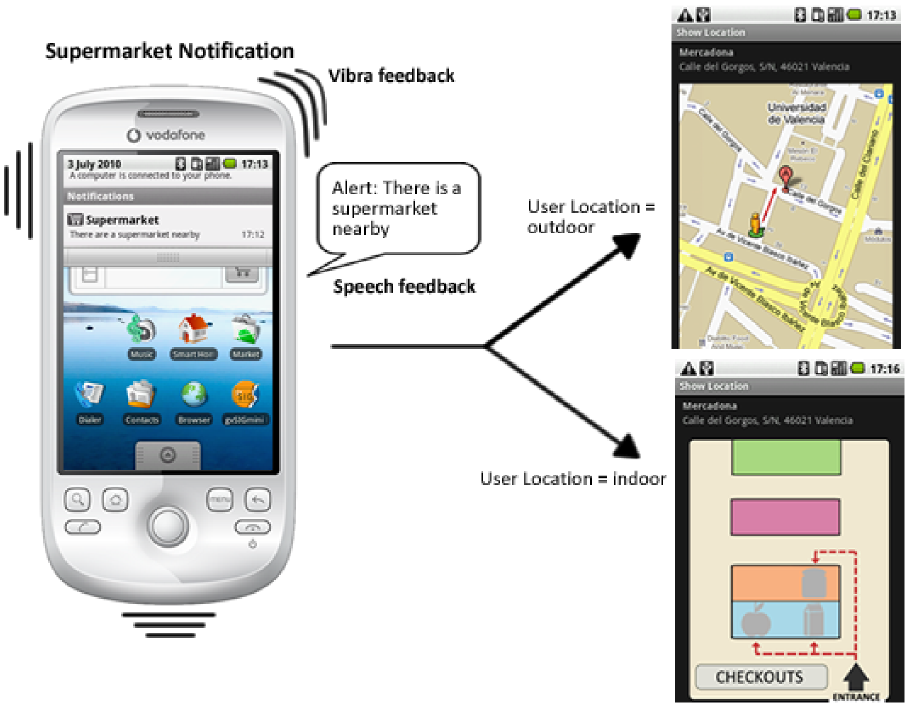


Figure 5.14: Android prototype

conjunction with the questionnaire defined by Vastenburg et al. (Vastenburg et al., 2009). On the one hand, IBM Post-Study is a questionnaire that measures user satisfaction with system usability. On the other hand, some questions were taken from the Vastenburg questionnaire to evaluate messages acceptability and interaction adaptation. Also, we included a NASA task load index (TLX)[7] test. This test assesses the user's subjective experience of the overall workload and the factors that contribute to it. These questionnaires can be found in Appendix B. An example of the application of the evaluation technique can be found in Chapter 8.

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

These questionnaires are the basis to detect change requirements and user dissatisfactions regarding the interaction obtrusiveness adaptation designed. Evaluating the questionnaires, designers refine the designed models in order to guarantee the user satisfaction with the system before move to the development phase. Also, several iterations of design-simulation can be made until the design meets the user requirements.

5.5 Discussion of our design method

In this section we introduce a discussion of the *usefulness* and *efficiency* of the proposed method. An evaluation of its usability for system designers can be found in Chapter 8.

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

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

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

Regarding the efficiency, the modeling solutions for interaction adaptation usually describe *what* information is presented to the user by means of an *Abstract User Interface*, and then define a discrete set of platforms, environments and user types to determine *how* the interaction will be offered for each set of context conditions (Calvary et al., 2003). But this discretization of context conditions presents some problems:

- *Similarities between the different context conditions are not exploited.* Context conditions are considered to be atomic without taking into account the existence of shared limitations and capabilities. For example, an auditory impaired user and a noisy environment both share the auditory limitation, so the interaction with the system would be more similar in these contexts compared to the interaction offered at other user. Another example is shown on the left of Fig. 5.15 between a mobile device and a PDA. Both have a limited screen, so the interaction with the system would be more similar in these devices compared to the interaction offered in a desktop computer or a device with no screen at all.

- *All combinations of context conditions are considered explicitly to define the interaction.* This implies specifying how interaction is derived from an *Abstract User Interface* for each platform-user-environment combination. For example, we should consider how to produce the interface for a visually-impaired user accessing the system from a mobile device platform in a noisy environment. Therefore, the complexity of interaction increases with the number of context conditions considered (see Fig. 5.15).

Since the promise of natural interaction of Ubiquitous Computing implies adapting to a large number of context conditions, we decompose the context conditions in their features (capabilities and limitations) represented as interaction aspects, and we use these features to describe the interaction in an abstract manner. Interaction features can

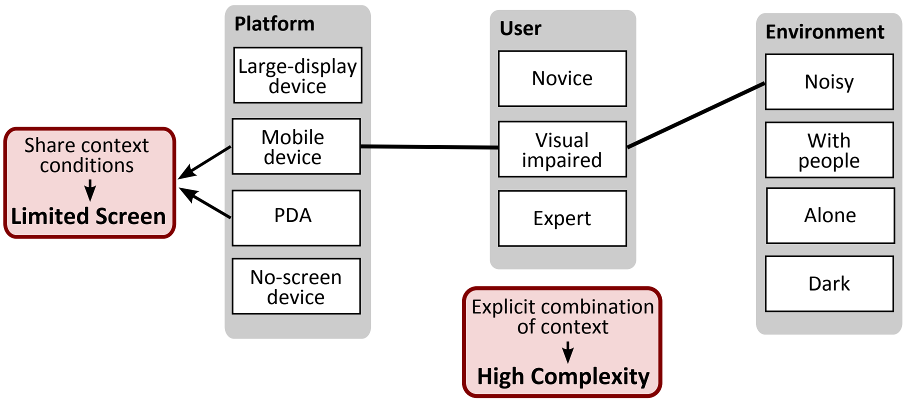


Figure 5.15: Problems of context condition discretization.

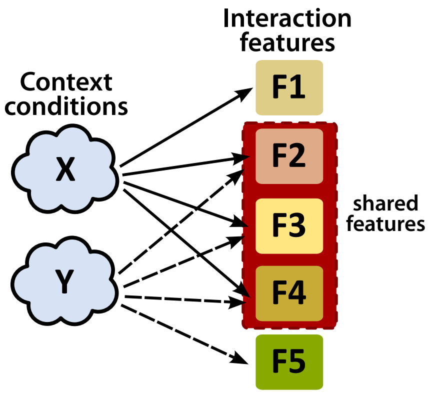


Figure 5.16: Context decomposition into features.

be shared among context conditions to indicate their commonalities. In this way, the above problems are solved. The features can be shared among context conditions to indicate their commonalities and differences. Figure 5.16 shows an example of the proposal. Context *X* has the features *F1*, *F2*, *F3* and *F4*, and the last three features (*F2*, *F3*, *F4*) are common to another context *Y*. We aim to express the interaction as a function of features. In this way, we could support contexts X and Y, and all another contexts that could be expressed as a combination of these shared features.

For example, a noisy context and a user with an auditory impairment require interaction not to be provided by means of audio. By considering the specification in terms of features the duplication of efforts in the development are minimized since both cases are expressed as the exclusion of the auditory feature. Avoiding the duplication of efforts in the development of services we guarantee the efficiency of the proposal.

5.6 Conclusions

This chapter introduces a design method to specify services according to the user needs and context conditions in terms of obtrusiveness without duplicating efforts in the development. When defining mobile interaction, many alternatives exist for defining the static structure of the user interface in a rapid manner (e.g., Visio stencils, paper prototypes). However, this only allows to validate the aspect of the UI in a particular moment. Our approach allows to visually represent how the interface changes according to different conditions. By analyzing the impact of different factors in the interface we can (1) produce specific variants of the application to target a particular device kind, and (2) define how the interface is adapted at runtime according to different user situations.

The designs defined according to the method can be easily put into practice. Fast prototyping techniques have been used to validate the adaptation for each service according to user satisfaction. The feedback obtained from the evaluation can be used to better adjust the models defined at design time.

Also, the design method provided relies on proven techniques and frameworks for context-aware modeling and implicit interaction design. The following chapter makes use of the models defined to provide a development method for the considerate services and adapt their interaction at runtime.

6 Self-Regulating Interactions Through Models at Runtime

Adapting obtrusiveness at runtime

Vision without implementation is hallucination.
—Benjamin Franklin (1706-1790).

According to the Considerate Computing vision (Gibbs, 2005), user attention is a primary resource to be considered by software systems. Although any kind of software system can benefit from being aware of the user attention level, this is especially relevant in the ubiquitous computing paradigm which promotes a natural interaction between the user and the environment (Weiser and Brown, 1997). Thus, ubiquitous services should behave in a considerate manner, demanding user attention only when it is actually required according to the user needs and context. To achieve this, interaction obtrusiveness of services should be adapted autonomously to the different ubiquitous and mobile devices that a user can possess (e.g., an Android/iOS-based mobile phone, tablets, etc.).

In Chapter 5, we presented a design method for capturing the degree of obtrusiveness required to interact with the ubiquitous services. By

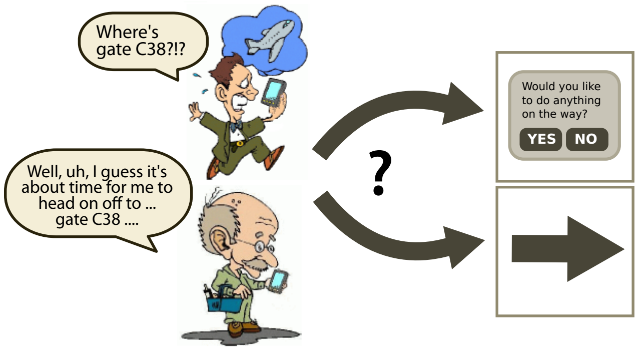

Where's gate C38?!?
Well, uh, I guess it's about time for me to head on off to ... gate C38 ....
?
Would you like to do anything on the way?
YES
NO

Figure 6.1: Example of how attentional resources of the user can call for different system responses.

considering the obtrusiveness aspects at design time, personalized services' interactions can be provided to avoid overwhelming users. However, the obtrusiveness level of services' interactions cannot be completely determined at design time and it must change during runtime according to changes in the user situation. This idea is illustrated in Figure 6.1. For example, consider a user in an airport to take a fly and thinking mainly about the task of moving swiftly through the airport terminal for not losing the fly because s/he arrives late. S/he probably prefers instructions that require the minimal attention (e.g., showing an arrow). By contrast, consider another user in the same airport with sufficient time to take the fly. This second user is more relaxed and can have a much greater attention (e.g., showing a dialog). In principle, the mobile service interaction should be appropriately adapted to each user situation automatically and seamlessly for the user.

In this chapter, we present a **self-regulating autonomic infrastructure** for adapting the degree of obtrusiveness required for each service automatically at runtime. This infrastructure exploits the design models at runtime to support the dynamic interaction obtrusiveness adaptation according to the user's situation (user's context). Our proposed approach has two main aspects:

Reuse of the design knowledge to achieve adaptation. We reuse the knowledge captured at design to describe the adaptation of the interaction in terms of obtrusiveness. In response to changes in the context, the system itself can query these models to determine the necessary modifications to its interaction components.

Reuse of existing model-management technologies at runtime. Since the models used are machine-processable and standard-based, current tools for model manipulation and traceability between models can be used to support our approach and propagate the changes in the obtrusiveness level into the interaction components.

In addition to design useful and considerate adaptations, an important prerequisite for the adoption of adaptive interactions is the development method (Höök, 2000). Before system deployment, the adaptive parts of services should be developed. We presented our model-based methodology to design adaptive service interactions in terms of obtrusiveness in Chapter 5. On this basis, and following the Model-Driven Engineering (MDE) principles (Schmidt, 2006), in this chapter we provide support to **automate the development of services** with adaptive interaction capabilities in terms of obtrusiveness. Thanks to MDE techniques, it has been possible to traverse the gap between the high-level concepts used at design and the technical details of the particular technology that is used for the services' implementation. Also, our approach reduces the development time because the developer does not have to implement all the components of the system and their adaptive behavior.

This chapter is structured as follows. Section 6.1 describes the way in which the development process is automated for the services' interactions defined. Section 6.2 presents the adaptation process that follows our approach to self-regulate the interaction obtrusiveness of services. Then, the autonomic infrastructure that supports the adaptation process is defined in Section 6.3. Section 6.4 describes the mechanisms used to adapt services' interactions into the managed devices. In Section 6.5, we specify how to evolve non-adaptive services into adaptive ones. Section 6.6 describes an experimentation to evaluate the scalability of the proposed infrastructure, and finally, Section 6.7 concludes the paper.

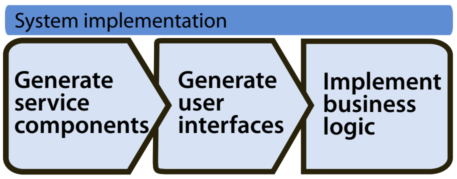


Figure 6.2: The different tasks in the implementation stage.

6.1 System implementation

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

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

Figure 6.2 shows the tasks carried out in this implementation stage. From the models obtained in the design phase, we can **generate the service components** of the different services and their adaptive **user intefaces**. Finally, the implementation is completed to include business logic of services. Also, from the developer perspective, several considerations must be taken into account when the service that is being developed requires adaptation capabilities such as the functionality to be adapted and the infrastructure required for the communication between the services and our autonomic infrastructure. This considerations are explained in Section 6.4.

The following subsections describe the mechanisms applied for obtaining the services' components and their user interfaces automatically.

6.1.1 Glue code generation

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

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

From the description of the system based on the defined metamodel, source code can be generated with model-to-text transformation techniques. Model-to-text generation tools provide mechanisms to traverse models and generate the code associated with them. We applied model-to-text transformations to formalize the development process defined in Chapter 5. Glue code generation has been implemented using XPand templates from the Model-to-Text (M2T) project[1], which is part of the Eclipse Modeling Project. The application of templates to models is similar to the way templates are used to generate dynamic web pages in the web application development area. Model elements can be iterated and pieces of code can be produced instantiating them with values obtained from the model. XPand is a statically-typed template language

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

with several features that simplify the code generation:

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

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

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

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

The following listing 6.1 shows a general structure of template files in order to introduce the basics of the XPand language. A template file consists of any number of *IMPORT* statements, followed by any number of *EXTENSION* statements, followed by one or more *DEFINE* blocks (called definitions).

Listing 6.1: General structure of a template file.

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

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

The basic statements used in the XPand language are:

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

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

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

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

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

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

The current implementation provides code-generation capabilities for two different aspects: (1) the components of the different services, including the *Android Manifest* and the different *Android classes* that are required for the implementation of the different components and (2) the different user interfaces to support the adaptations. A detailed description of the artifacts generated is provided below.

Service components generation

The services defined in our system can be composed by different components defined in the current or external systems. Thus, we propose to define the way the components are integrated with the other components of the system in order to (1) explicitly state their rationale, (2) use this knowledge for automating the development, and (3) detect the sources of context information that can trigger an interaction adaptation.

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

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

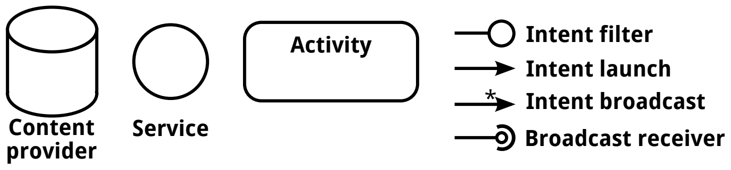


Figure 6.3: Graphical notation used to represent components of the Android application framework

are available today including mobile phones, netbooks, e-book readers or TVs.

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

Figure 6.3 illustrates the notation used to describe Android components. The main components from the Android application framework are represented in a graphical manner. The notation is aligned with other common notations such as Business Process Management Notation (BPMN) (OMG, 2006b) or the Unified Modeling Language (UML) (Rumbaugh et al., 1998b) for the sake of intuitiveness.

The intent-based communication mechanism among components is also represented in our notation (see Fig. 6.3, right) to indicate their possibilities for the components to interact. The way in which compo-

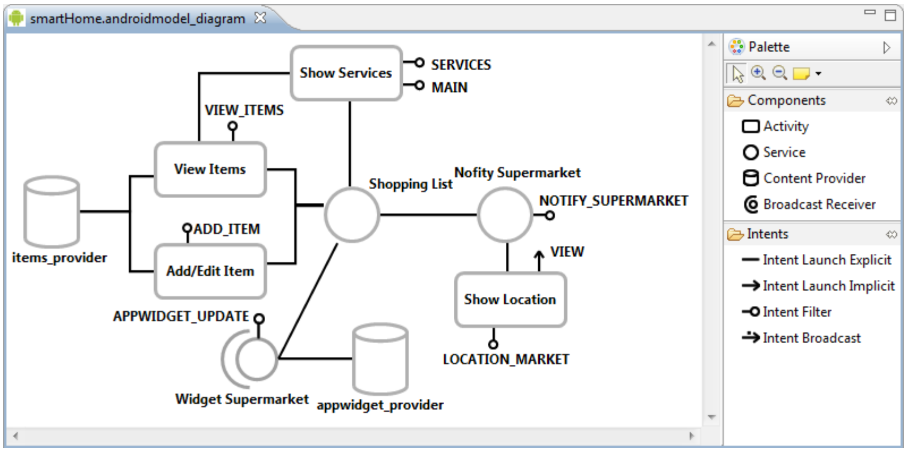


Figure 6.4: Service components model

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

Figure 6.4 shows the model for the components of a shopping list and supermarket notification services using the notation introduced. The system is composed by four activities corresponding to the user interfaces provided. These activities have defined the intent filters associated to the actions they can perform such as *ADD_ITEM* or *VIEW_ITEMS*. *Show Location* activity launches the intent *VIEW* to show the map of the location. Moreover, the *Show Services* activity has the intent filter *MAIN* to mark this activity as the initial activity. There are two content providers: one for offering the items of the shopping list and another for offering the information to update the *Widget Supermarket receiver*. There are also two services in the system: the *Shopping List* service in charge of orchestrating the communication between the components and the *Notify Supermarket* service in charge of launching a notification.

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

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

From this service components model, developers can **generate the Android components of the whole services** defined. Figure 6.5 describes the elements that are produced by the transformation from the model.

Listing 6.2 shows the code of the main template that is used for orchestrating the generation of all the components. The excerpt of the transformation declares the *main* rule by means of the *DEFINE* keyword. The *main* transformation rule is used to generate the components

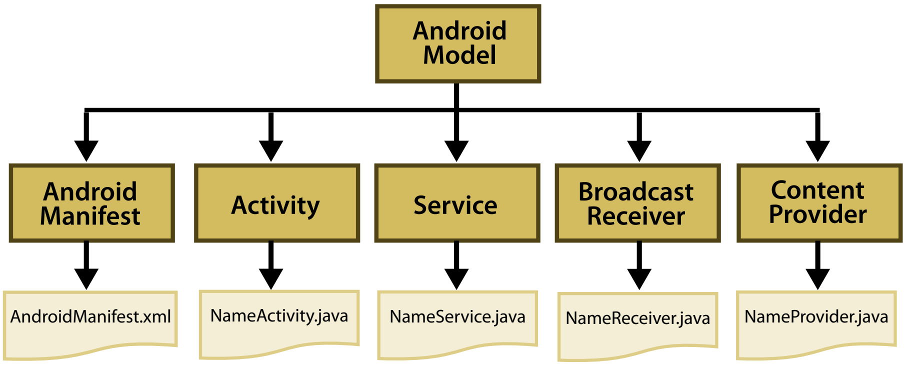


Figure 6.5: Global schema of the elements generated by the transformation.

of the application. This rule is applied to the whole system and it is defined for the *App* element of the Android components metamodel (see the metamodel in Appendix A). By means of the *EXPAND* command, the initial structure of the application, the *Android Manifest* file and the rest of Android components are expanded for the generation.

Listing 6.2: Excerpt of the code generation template that produces the calls to all the components of an Android application.

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

Listing 6.3 shows the code of one of the templates that is used for generating the *Android Manifest*. The excerpt of the transformation declares the *manifest* rule. This rule is used to generate the fragment of the *Android Manifest* that is associated to each component. In the example, we show that this rule is also applied to the whole system since it is defined for the *App* element of the Android components metamodel. Generation rules control the creation of new files (e.g., source code, configuration files, resource descriptions, etc.) and the generation of their correspondent content. The *FILE* statement defines the output file for the code generation (in the example, an *AndroidManifest.xml* file is generated into a folder named after the *App*).

Listing 6.3: Excerpt of the code generation template that produces the Android Manifest file.

```
IMPORT AndroidModel
EXTENSION template::GeneratorExtensions
DEFINE manifest FOR App
FILE name + "/AndroidManifest.xml"
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res
    /android"
      package="generatePackageName(name)"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:label="@string/app\_name">
        EXPAND activitiesDeclaration FOREACH activities()
        EXPAND servicesDeclaration FOREACH services()
        EXPAND broadcastReceiversDeclaration FOREACH
            broadcastReceivers()
        EXPAND providersDeclaration (name) FOREACH
            providers()
    </application>
  <uses-sdk android:minSdkVersion="3" />
</manifest>
ENDFILE
ENDDEFINE
```

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

The generation of the rest of the *Android Manifest* is carried out by different rules, each one for a specific kind of component. For example, *activitiesDeclaration* rule is applied to *Activity* components for the generation of the activity statements. Listing 6.4 illustrates the definition of the *activitiesDeclaration* rule applied to this kind of components. This rule declares the activity in the manifest and their intent filters according to the model defined. For each intent filter declared in the model, the rule adds an intent filter named with the name declared in the model. If the component has the Main intent filter, the default category *android.intent.category.LAUNCHER* is also added. This determines that the activity can be launched by the user directly. The template of this example makes use of conditional statements (see the IF, ENDIF instructions) to add the default category for the generated intent.

Listing 6.4: Generation rule that produces the Android Manifest fragment corresponding to Activities.

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

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

The code generation supported in this part automates the definition of the **Android Manifest** and the **Java classes** that are required for the implementation of the different service components according to the service components model. **Intent processing code** is also generated. Although full code generation is not provided for component implementation, the provided code skeletons let developers focus on the implementation of the business logic behavior, avoiding to deal with particular details of the target technology. Since the Android specific artifacts are generated, the use of the Android application framework is made transparent to the developer, who only has to deal with Java programming.

User interface generation

We also propose to generate the user interfaces to support the interaction adaptation for the different user situations. For this generation, we make use of the design models defined in Chapter 5.

As we introduced in Chapter 5, Android platform has a node-based user interface. In particular, in an Android application, the user interface is built using *View* and *ViewGroup* objects (see Fig. 6.6). *View* objects are the basic units of user interface expression on the Android platform that serves as the base for subclasses such as *widgets*. The *ViewGroup* class serves as the base for subclasses called *layouts* which offer different kinds of layout architecture, like linear, tabular and relative. A *View* object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen. A *View* object handles its own measurement, layout, drawing, focus change, scrolling, and key/gesture interactions for the rectangular area of the screen in which it resides. As an object in the user interface,

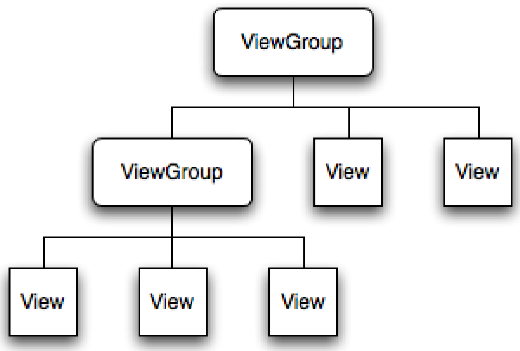


Figure 6.6: Hierarchy for defining Android UIs

a *View* is also a point of interaction for the user and the receiver of the interaction events.

Thus, on the Android platform, the user interface is defined using a hierarchy of *View* and *ViewGroup* nodes, as shown in the Figure 6.6. The most common way to define a user interface expressing the view hierarchy is with an **XML layout file**. XML offers a human-readable structure for the layout, much like HTML. Each element in XML is either a *View* or *ViewGroup* object (or descendant thereof). *View* objects are leaves in the tree and *ViewGroup* objects are branches in the tree.

In this way, for the implementation of the user interfaces, we produce the Android *XML layout file* for all the user interfaces of the whole system from the concrete interaction components model. Specifically, in each generation iteration, we produce the user interfaces of an specific interaction configuration.

The process for generating the code of a service interface takes as input the concrete interaction component model configured with the active nodes of the service for a specific configuration. The transformation template is composed by different rules that perform the generation. Listing 6.5 shows an excerpt of the template used for generating the XML layout file. The transformation template is composed by different rules that perform the generation. These rules are explained below:

Listing 6.5: Excerpt of the code generation template that produces the XML layout for each user interface.

```
DEFINE Root FOR FeatureModelPackage::FeatureModel
 FILE Name+".xml"
  <?xml version="1.0" encoding="utf-8"?>
 EXPAND FindRoot FOREACH Features
 ENDFILE
ENDDEFINE
DEFINE FindRoot FOR FeatureModelPackage::Feature
 FOREACH Attributes AS e
   IF e.Name=="root" && e.Value=="true"
     EXPAND PrintFeature FOR this
   ENDIF
 ENDFOREACH
ENDDEFINE
DEFINE PrintFeature FOR Feature
< Name
 FOREACH Attributes AS e
   IF e.Name == "root"
    xmlns:android="http://schemas.android.com/apk/res/
        android"
   ELSEIF
    android: e.Name =" e.Value "
   ENDIF
 ENDFOREACH
>
 FOREACH CardinalityBased_Relationships.typeSelect(
     Mandatory) AS e1
   EXPAND PrintFeature FOR e1.To
 ENDFOREACH
 IF ! CardinalityBased_Relationships.typeSelect(
     Alternative).isEmpty
   FOREACH CardinalityBased_Relationships.typeSelect(
       Optional) AS e1
     EXPAND PrintFeatureAlternative FOR e1.To
   ENDFOREACH
 ELSE
 FOREACH CardinalityBased_Relationships.typeSelect(
     Optional) AS e1
   EXPAND PrintFeatureOptional FOR e1.To
 ENDFOREACH
```

```
      ENDIF
     </ Name >
   ENDDEFINE
```

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

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

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

The result is an Android XML layout file of the service interface. Some XML attributes of the components needed for generating the layout are defined as node attributes in the concrete interface model. In this way, nodes are generated with their appropriate attributes.

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

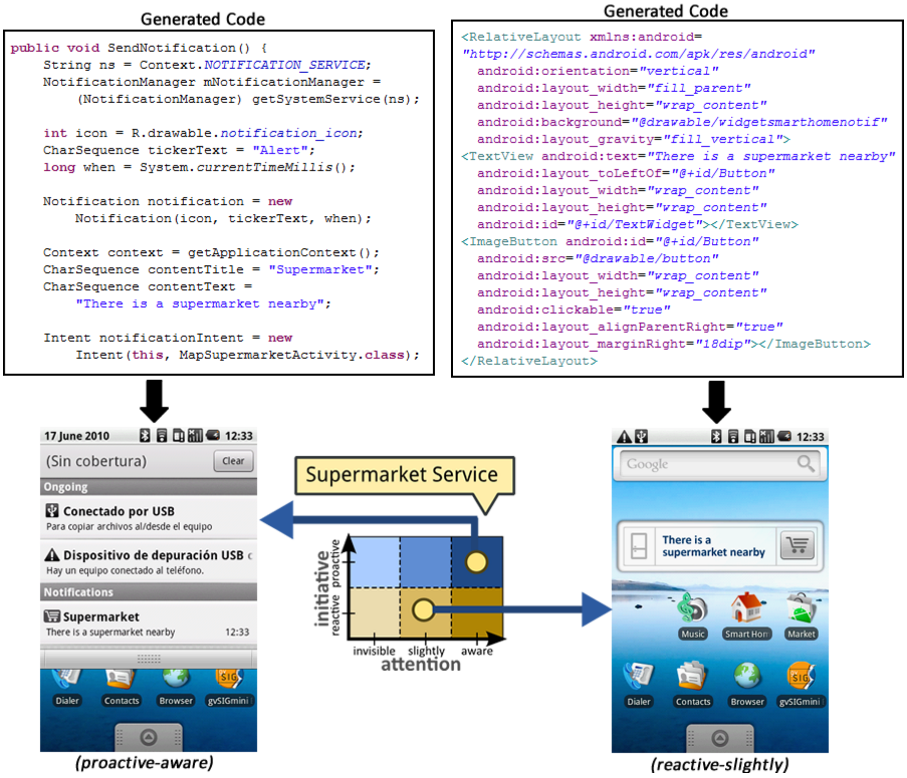
However, some interface components can not be implemented by means of the *XML layout file*, such as the notifications, the use of specific libraries for the physical interaction, etc. In particular, we define *Android services* for implementing these components. So, we have defined another templates for generate these specific components.

For example, notifications are generated by means of this template because they are initiated from a *Service*. In this way, the notification can be created from the background, while the user is using another application. This is implemented checking the name of the node in the *PrintFeature* rule.

Figure 6.7 shows the two kinds of generation for the supermarket service in two different obtrusiveness levels. On the one hand, for the (*aware, proactive*) obtrusiveness level, the service is presented by means of a *status bar notification* (through all the design process described in the previous chapter). An excerpt of the generated code for the *status bar notification* and the rendering of this code is shown at the left of the Figure 6.7. On the other hand, for the (*slightly, reactive*) obtrusiveness level, a *widget* is used and the generated code is shown at the right of the Figure 6.7. In this way, the services are generated and adapted for each user situation.

6.2 The self-regulating system

The design models capture the knowledge required to provide considerate services' interactions. However, in order to allow an autonomic and unobtrusive adaptations, the ubiquitous services must themself regulate their interactions with users according to the different user situations. Specifically, we argue that a service can use the appropriate interaction components dynamically at runtime according to the user situation. The *unobtrusive adaptation space* specifies the different ways in which services can interact with users. *Interaction features* specify the possible interaction configurations that support the services for each obtrusiveness level, while an adaptive user interface made by *concrete interaction*


Generated Code

```
public void SendNotification() {
    String ns = Context.NOTIFICATION_SERVICE;
    NotificationManager mNotificationManager =
        (NotificationManager) getSystemService(ns);

    int icon = R.drawable.notification_icon;
    CharSequence tickerText = "Alert";
    long when = System.currentTimeMillis();

    Notification notification = new
        Notification(icon, tickerText, when);

    Context context = getApplicationContext();
    CharSequence contentTitle = "Supermarket";
    CharSequence contentText =
        "There is a supermarket nearby";

    Intent notificationIntent = new
        Intent(this, MapSupermarketActivity.class);
```

Generated Code

```
<RelativeLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:background="@drawable/widgetsmarthomenotif"
  android:layout_gravity="fill_vertical">
<TextView android:text="There is a supermarket nearby"
  android:layout_toLeftOf="@+id/Button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/TextWidget"></TextView>
<ImageButton android:id="@+id/Button"
  android:src="@drawable/button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:clickable="true"
  android:layout_alignParentRight="true"
  android:layout_marginRight="18dip"></ImageButton>
</RelativeLayout>
```

17 June 2010 12:33
(Sin cobertura) Clear
Ongoing
Conectado por USB
Para copiar archivos al/desde el equipo
Dispositivo de depuración USB c
Hay un equipo conectado al teléfono.
Notifications
Supermarket
There is a supermarket nearby 12:33

Figure 6.7: Different generations of the same service

components (components of the target device) can be rapidly retargeted to a specific configuration (a set of interaction modalities) in response to user situation changes (triggered by a context change).

Specifically, our approach follows a dynamic computer-human interaction loop that has a goal to regulate (i.e., a self-regulating system) (Dubberly et al., 2009) (see Fig. 6.8(a)). In our case, this goal is to provide considerate services' interactions. Thus, the process that follows our system is to *gather* information about user's context, *analyze* this information to check the user's situation, and *adapt* the interaction

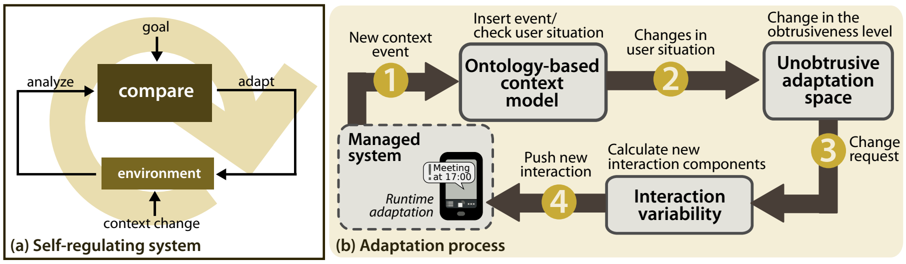

goal
analyze
compare
adapt
environment
context change
(a) Self-regulating system
New context event
Insert event/ check user situation
Ontology-based context model
Changes in user situation
Change in the obtrusiveness level
Unobtrusive adaptation space
Change request
Managed system
Meeting at 17:00
Runtime adaptation
Push new interaction
Calculate new interaction components
Interaction variability
(b) Adaptation process

Figure 6.8: (a) Self-regulating system and (b) adaptation process.

of the services according to the *system's goal* (not to disturb the user) by *comparing* the last and the new user situation. To achieve this, our approach follows the adaptation process depicted in Figure 6.8(b):

1. **Monitoring 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 situation is analyzed to check whether the user's current situation has changed and some adaptation needs to be made (e.g., user from a relaxed situation to be in a hurry). This inference is based on logic rules that uses the context information updated in the ontology. For example, a change in the current time could mean that the user is in a hurry.

2. **Analyzing the attentional demand.** The second step of the adaptation process is triggered when there is a change in the user situation. 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 situation. For example, the *UserInAHurry*, a new user situation, can trigger a change in the obtrusiveness level demanding less attention. A change request with the new obtrusiveness level is generated.

3. **Comparing the interaction mechanisms.** The third step of

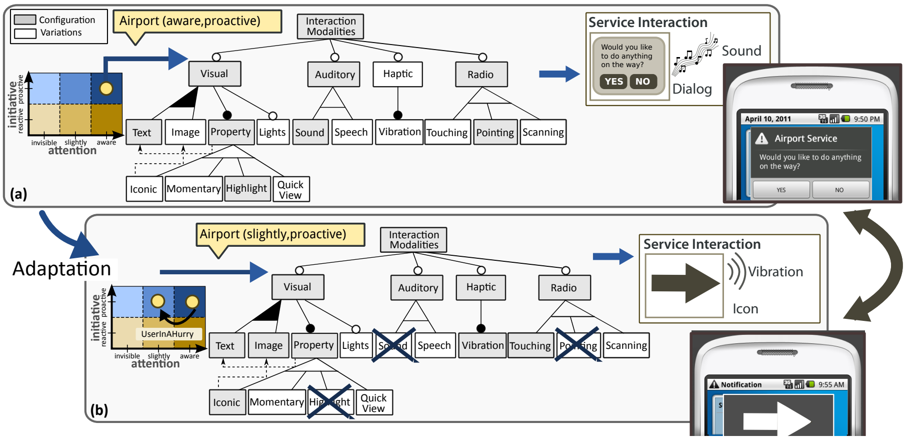


Figure 6.9: Runtime adaptation example.

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 features are calculated in terms of *interaction increments* (interaction mechanisms to activate) and *interaction decrements* (interaction mechanisms to deactivate). 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.

In order to specify which interaction features support a certain service for a given obtrusiveness level, the *superimposition operator* ($\odot$) is defined. The superimposition operator takes a service and an obtrusiveness level and returns the set of interaction features required for the service. Some examples of the relationship between the obtrusiveness level for the *airport* service and its mapping with the interaction features (see Fig. 6.9) are as follows:

$$\odot Airport(aware, proactive) = \\ \{visual, property\_highlight, text, auditory, sound, radio, pointing\} \\ \odot Airport(slightly, proactive) = \\ \{visual, property\_iconic, text, image, haptic, vibration, radio, touching\}$$

The results of the *InteractionIncrement/Decrement* operations given the adaptation of the airport service when the user is in a hurry (*UserInAHurry* situation) are as follows (see Figure 6.9):

$$\begin{aligned} InteractionIncrement_{UserInAHurry} =& \{property\_iconic, image, haptic, \\ & vibration, touching\} \\ InteractionDecrement_{UserInAHurry} =& \{property\_highlight, auditory, \\ & sound, pointing\} \end{aligned}$$

These operations indicate how the services should adapt their interaction in order to move from one interaction configuration to another. Fig. 6.9 illustrates an example of the adaptation strategy for the airport service. In this figure, the service first is in the *(aware, proactive)* level by means of a *highlighted* visual property with *sound* feedback and the *pointing* interaction technique are used to provide the interaction. But, when the mobile device senses the user is in a hurry (due to a context change), the interaction is adapted to demand less attention of the user. With this adaptation, the mentioned interaction mechanisms are no longer used for the interaction with this service; instead, an *iconic* visual property with *vibration* and the *touching* interaction technique are used (as calculated by the operations).

4. **Adapting the interaction components.** When the new interaction features are calculated according to the user situation, the system applies the adaptation in the ubiquitous 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 features into the concrete interaction mechanisms of each platform. This is done by querying the weaving

model to realize the mappings between the features and their related interaction components.

For example, if our system indicates that an icon with vibration is needed for service S, this operation replaces these components to the ones that match with the specific target technology (e.g., in an Android system, the icon is translated into the status bar component). For the results obtained when trigger the user situation *UserInAHurry* shown above, the following concrete interaction components are calculated:

$$Increments = \{GroupNotif, StatusBar, Icon, Text, Vibration\}$$
$$Decrements = \{GroupDialog, GroupAudio, Sound\}$$

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

6.3 AdaptIO: an infrastructure for adapting interaction obtrusiveness

In order to carry out the adaptation process, we provide an autonomic infrastructure (AdaptIO) that enhances services with interaction obtrusiveness adaptation capabilities. Our infrastructure provides the following benefits:

- It is available to the ubiquitous 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 to adapt interaction based on user attention in a technology-independent manner.

Next, we first explain the infrastructure at the conceptual level, and then, we explain how we have developed it giving technological details.

6.3.1 The Autonomic Infrastructure

In order to make our user-centered obtrusiveness adaptation a reality, we define a model-based infrastructure, i.e., AdaptIO. AdaptIO is an autonomic infrastructure to support the considerate interaction adaptation of services in the ubiquitous computing domain. To achieve this, it senses the context from environmental and mobile sensors and adapts the interaction resources of each service in terms of obtrusiveness. Its autonomic behavior is based on the design models introduced in Chapter 5. 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. Also, we provide a set of components in a pluggable manner for addressing the technological heterogeneity of ubiquitous and mobile devices.

Fig. 6.10 illustrates the AdaptIO infrastructure components and their connection to context sources, services and managed devices. 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 conform the system. For achieving the self-adaptation, our infrastructure is based on the IBM reference model for self-management (IBM, 2006), which is called the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) loop. In our case, the knowledge is defined in the high-level models. As shown in Fig. 6.10, the process is composed by four components that share the models.

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 Situation Analyzer*) using the design models. Figure 6.11 describes the over-

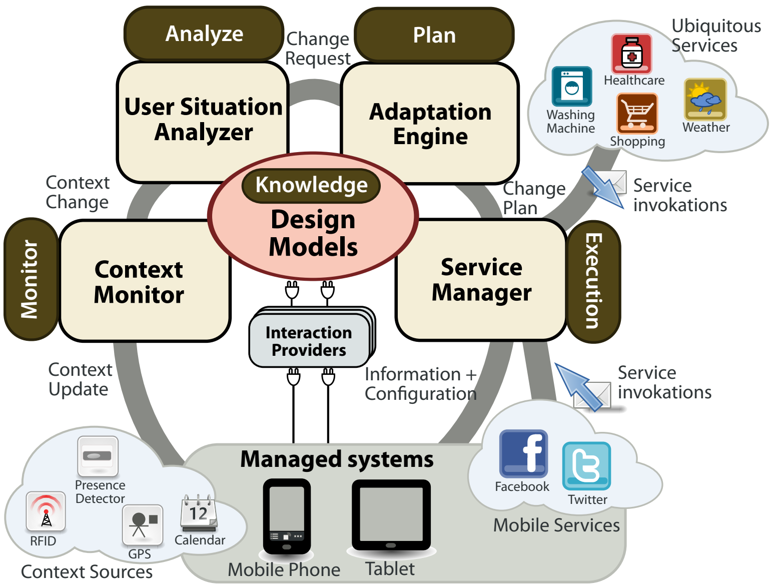


Figure 6.10: Infrastructure components overview

all adaptation flow among these components. In the next subsections, more details about these components and their supported operations are given (with some implementation details).

Context Monitor

The *Context Monitor* is the component 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 Situation Analyzer*.

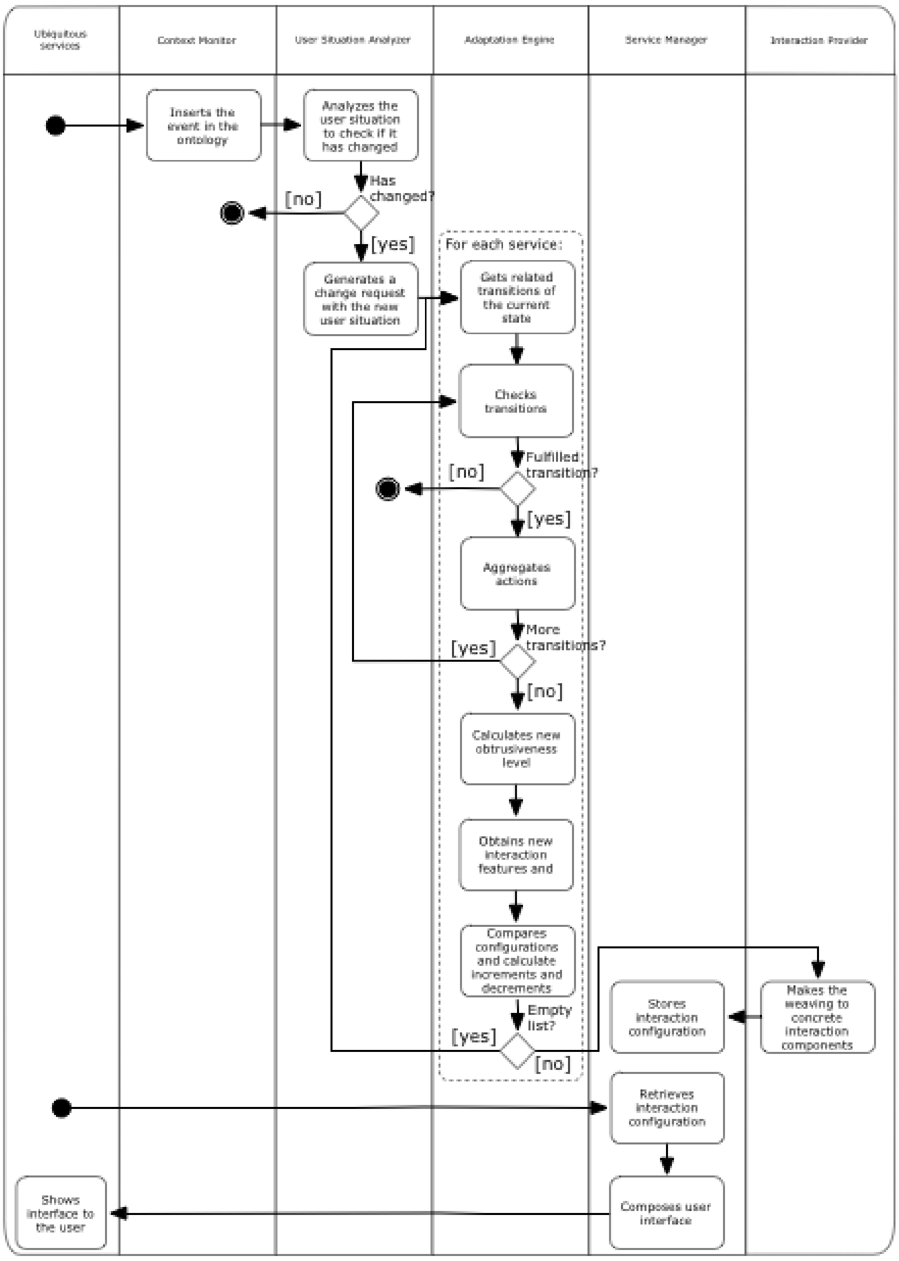

Ubiquitous services
Context Monitor
User Situation Analyzer
Adaptation Engine
Service Manager
Interaction Provider
Inserts the event in the ontology
Analyzes the user situation to check if it has changed
Has changed?
[no]
[yes]
Generates a change request with the new user situation
For each service:
Gets related transitions of the current state
Checks transitions
Fulfilled transition?
[no]
[yes]
Aggregates actions
More transitions?
[yes]
[no]
Calculates new obtrusiveness level
Obtains new interaction features and
Compares configurations and calculate increments and decrements
Empty list?
[yes]
[no]
Makes the weaving to concrete interaction components
Stores interaction configuration
Retrieves interaction configuration
Composes user interface
Shows interface to the user

Figure 6.11: Adaptation flow of the AdaptIO behavior

Context changes are physically detected by environmental (physical sensors) and mobile sensors (virtual sensors), which are controlled by ubiquitous and mobile services. Thus, in order to capture context changes, the monitor is continuously monitoring the execution of the services. Specifically, the pervasive services that control the physical sensors are developed using the development method presented in (Serral et al., 2010), which allows us to automatically generate Java/OSGi[2] pervasive services from high-level abstraction models. Using OSGi, the *Context Monitor* can listen 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 device that monitor changes in their own device 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.

To implement the operation to insert context events in the ontology (*InsertContextEvent* operation), we use the INSERT form of SPARQL[3]. 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 which 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 as the one we use for the context model. For example, Listing 6.6 shows the SPARQL expression to set that a given user is at home.

Listing 6.6: Example of *InsertContextEvent* operation.

```
DELETE DATA FROM <http://www.pros.com/Person>
{<http://www.pros.com/Bob> pros:personLocatedIn  ?value}

INSERT DATA INTO  <http://www.pros.com/Person>
{<http://www.pros.com/Bob>  pros:personLocatedIn  "pros:
    Home"}
```

[2]http://www.osgi.org/
[3]http://www.w3.org/TR/rdf-sparql-query/

User Situation Analyzer

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

In order to accurately infer the user's situation, it is based on logic rules. These logic rules use the context information updated in the context model. For example, Listing 6.7 shows the rule to infer when the user is with company:

Listing 6.7: Example of the *withCompany* rule.

```
[withCompany: (?user rdf:type pros:Person)
      (?user pros:usersInLocation ?location)
      (?person1 rdf:type pros:Person)
      (?user pros:knows ?person1)
      (?person1 pros:usersInLocation ?location)
      (?user pros:socialRelationships ?person1)
      ->
      (?user pros:currentSituation pros:withCompany)]
```

To implement the rules, we have used the Jena Framework[4]. Jena is a Java framework for building Semantic Web applications that provides a programmatic environment for OWL and SPARQL and includes a rule-based inference engine. We have a rule repository that contains a set of logic rules. In the design phase, rules are manually added in the rule repository by designers. However, end-users can also update these rules and define his/her own situations at runtime by means of the *User Specification Interface* (explained in Chapter 7).

The main operations that support the *User Situation 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 situation

[4]http://jena.apache.org

changes in the ontology. Listing 6.8 shows an excerpt of the main class that call the operations of the *User Situation Analyzer*.

Listing 6.8: Excerpt of the main class of the User Situation Analyzer.

```java
public class UserSituationAnalyzer {
  private static Logger logger;

  public static void main(String[] args) {
    initFactoriesAndPaths(args);

      RuleEngine engine = RuleEngineFactory.getInstance().
          createRuleEngine(
          RuleEngineFactory.BUILTIN);
      engine.loadRulesFromFolder("rules");

      RdfGraph context = RdfGraphFactory.getFactory().
          createRdfGraph(
          RdfGraphFactory.QUERYABLE);
      context.retrieveFromFile("AdaptIO-ContextModel.owl");

      engine.inferNewData(context);
      RdfGraph data = engine.getInferredData();
  }
}
```

Adaptation Engine

When alerted by the *User Situation 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 situation. If so, the engine calculates the necessary modifications to the interaction features 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 6.2), the adaptation engine queries the obtrusiveness and feature model at runtime. To do this, it is based on four main operations: (1) the *getTransitionsToCheck* operation, (2) the *checkUserSituation* operation, (3) the *triggerTransition* operation, and (4) the *getDifferences* operation.

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

- For each service, it gets the output transitions of the current obtrusiveness level of the service. This activity is supported by the *getTransitionsToCheck* operation.
- Then, it checks the transitions to see if any user situation is fulfilled according to the user's current situation (it means that an adaptation needs to be made). This is supported by the *checkUserSituation* operation.
- For the transitions fulfilled, it aggregates their actions to calculate the target obtrusiveness level and avoid inconsistencies when trigger the transitions.
- 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 *Adaptation Engine* is implemented in Java/OSGi and uses Eclipse Model Query[5] (EMFMQ) to query the models at runtime and the EMF compare plugin[6] 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 use these queries to search for and get the instances of the model that need to be accessed or modified. EMF Compare allows model comparison to the EMF framework, providing a

[5]http://www.eclipse.org/modeling/emf/?project=query
[6]http://www.eclipse.org/emf/compare/

generic support for any kind of metamodel in order to compare models. Specifically, we use the *diff* class to calculate the difference between the two interaction configurations.

Listing 6.9 shows the implementation in Java of the *getTransitionsToCheck* operation. It creates a list with the transitions of the current obtrusiveness level to check.

Listing 6.9: Implementation of the *getTransitionsToCheck* operation.

```java
public ArrayList<Transition> getTransitionsToCheck() {
    this.machineChanged=false;
    ArrayList<Transition> transitionsToCheck=new ArrayList<
        Transition>();
    EList<Transition> transitions=this.getCurrentState().
        getStateTransitions();
    for (Transition transition : transitions) {
        transitionsToCheck.add(transition);
    }
    return transitionsToCheck;
}
```

Listing 6.10 shows the code of the *checkUserSituation* operation which checks if the user situation of the transition is fulfilled.

Listing 6.10: Implementation of the *checkUserSituation* operation.

```java
public boolean checkUserSituation(String userSituation) {
    boolean result = false;
    result = ManageOntology.queryPelletAsk(userSituation);
    return result;
}
```

Once the user situation of the transition is fulfilled, the transition is triggered in order to calculate the interaction modifications (differences). We have implemented the *triggerTransition* operation (see Listing 6.11) by retrieving the current interaction configuration and applying the transition over this configuration. This means, we process the transition to get the action associated to it (i.e., the target obtrusiveness level) and obtain the configuration of the target obtrusiveness level.

Listing 6.11: Excerpt of the implementation of the *triggerTransition* operation.

```
public ConfigurationModel applyTransition(
    ConfigurationModel cmToEvolve, Transition
    transitionToAply){
    ConfigurationModel cmEvolved=cmToEvolve;
    ArrayList<Feature> presentFeatures=new ArrayList<
        Feature>();
    for (FeatureState featureState : cmToEvolve.
        getFeatureStates()) {
        presentFeatures.add(featureState.getFeature());
    }
    EList <Action> actionsToAply=transitionToAply.
        getResolution().getActions();
    for (Action action : actionsToAply) {
        if(presentFeatures.contains(action.getFeature())){
            cmEvolved=updateFeatureState(cmEvolved, action.
                getFeature(), action.getActionType());
        }
        else{
            cmEvolved=createNewFeatureState(cmEvolved,
                action.getFeature(), action.getActionType()
                );
        }
    }
    return cmEvolved;
}
```

Finally, with the two configurations, we calculate the differences between them in order to obtain the increments and decrements of the interaction features (see Listing 6.12).

Listing 6.12: Excerpt of the implementation of the *getDifferences* operation.

```
if(smManager.hasTheMachineChanged()){
    DiffModel differencesModel = DiffService.doDiff(
        MatchService.doMatch((EObject)smManager.
        getLastConfiguration(),(EObject)smManager.
        getCurrentConfiguration(), Collections.<String,
        Object>emptyMap()), false);
    FeatureModelConfigurationChanges thisSMChanges=
        getDifferences(differencesModel);
    thisSMChanges.setStateMachineModelName(smManager.
        getStateMachineName());
```

```
      thisSMChanges.setCurrentStateName(smManager.
          getCurrentState().getName());
      this.systemChanges.add(thisSMChanges);
  }
```

This plan of high-level interaction increments/decrements is notified to all the installed *Interaction Providers*. As the adaptation is performed in a technology-independent manner (high-level concepts), we need a mechanism to translate the abstract interaction resources to use into the concrete ones of each platform. To make this, we define the *Interaction Providers*, pluggable components in charge of converting the high-level interaction plan into a concrete one with the specific interaction components of the underlying managed systems. These components query the concrete interaction component model and the weaving model. For example, the *Banner* is translated to a *Status Bar* on Android and a *Banner* on iOS. Finally, the plan is passed to the *Service Manager*. The operation in charge of supporting this translation is the *doWeaving* operation. Listing 6.13 shows the implementation of this operation.

Listing 6.13: Excerpt of the implementation of the *doWeaving* operation.

```
  private TargetModelChanges doWeaving(
      FeatureModelConfigurationChanges configurationChanges){
      TargetModelChanges temp=new TargetModelChanges();
      this.loadModelURIs();

      RunTimeModel rtm=new RunTimeModelImpl(sourceModelURI,
          sourceModelPackage, weavingModelURI, targetModelURI
          , targetModelPackage);

      ArrayList<EObject> aux=new ArrayList<EObject>();
      for (Feature featAux : configurationChanges.
          getAdditions()) {
        aux.add((EObject)featAux);
      }
      temp.setIncrements(rtm.getTargetModelElements(aux));
      aux=new ArrayList<EObject>();
      for (Feature featAux : configurationChanges.getRemovals
          ()) {
        aux.add((EObject)featAux);
```

```
    }
    temp.setDecrements(rtm.getTargetModelElements(aux));
    return temp;
  }
```

Service Manager

The *Service Manager* is the component in charge of applying the adaptation to the underlying system. When it receives a change plan of the interaction components to adapt 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 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 useful for the user, etc.), the *Service Manager* retrieves the interaction components to use 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 services' information according to 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. For receiving data from the services, a RESTful web service has been implemented. In this way, services can send any kind of data to the *Service Manager* only by sending the information via HTTP. The Restlet Framework[7] has been used for its implementation. The information has been formatted in JSON, a lightweight data-interchange format. In this way, all the components understand the format of the information.

Execute a service adaptation. Once a service invocation is received, it is adapted to the user's current situation and sent to the *Managed Systems*. 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 the registered devices. Specifically, we have used Android Cloud to Device Messaging[8] (C2DM) for the Android platform and Apple Push Notification service (APNs) for iOS. Listing 6.14 shows an excerpt of the code to send the information to an Android-based mobile device.

Listing 6.14: Excerpt of the implementation to use the C2DM service.

```
$message = json_encode($not);
$handler = fopen("registerID.txt","r");
$registrationID = fgets($handler);
fclose($handler);

$tooken = googleAuthenticate("username", "83055294",
    $source="Company-AppName-Version", $service="ac2dm
    ");
$re=sendMessageToPhone($tooken, $registrationID, "
    collapsed", json_encode($not));
```

6.4 Deployment of the infrastructure

[7]http://www.restlet.org/

[8]https://developers.google.com/android/c2dm/

This section provides details about the architecture followed to deploy our infrastructure, describing the key design choices and solutions. AdaptIO follows a client/server architecture. The context processing, user situation inferring, adaptations and services' 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. In this way, the logic remains on the server side avoiding mobile clients to deal with complex processes that may have an impact on performance.

6.4.1 Server Side Subsystem

The components of the server side (*Context Monitor*, *User Situation Analyzer*, and *Adaptation Engine*) are implemented using Java/OSGi technology. Using this technology, we achieve that the autonomic infrastructure is an operative system independent and can be dynamically constructed from reusable and collaborative components, which are known in the OSGi terminology as *bundles*. Thus, the infrastructure is developed to be run in 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. This registration makes the services discoverable through the registry so that other bundles can use them. Thus, when a bundle queries the registry, it obtains references to actual service objects 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 to add new devices (*Managed Systems*).

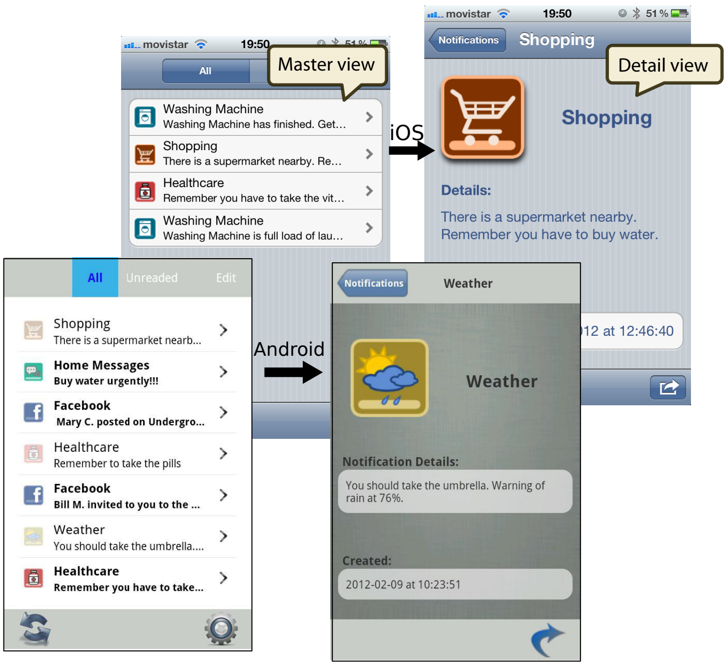
The framework also manages dependencies among services to facilitate coordination among them. These dependencies are implemented by using *Wire* objects. A Wire object acts like a communication channel between a Producer service and a Consumer service. When a wire is created, the Producer service can produce information to be used by the Consumer service. Thus, we use the *Wire Admin Service* in OSGi to address the inter-component eventing mechanism between the components of our system such as the *Context Monitor* and the *User Situation Analyzer*, this last and the *Adaptation Engine* or this last and the *Interaction Providers*. When a component has an event to deliver, the component source calls all 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 ubiquitous environments by means of the service discovery. Services expose external devices, such as UPnP devices or KNX-EIB devices which are the ones that we use to gather context information besides mobile sensors.

Finally, the *Service Manager* is implemented in PHP, a server-side scripting language interpreted by an Apache web server and connected to a MySQL database that stores the received interaction configurations. In this way, we keep track of the adaptation traces to be able to analyze it later and check the proper functioning of the autonomic infrastructure. Also, these traces are shown in a web page to assist engineers in the task of checking the correct adaptations of the services.

6.4.2 Client side subsystem: managed systems



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
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

Figure 6.12: Implementation of the managed system on iOS and Android.

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 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 centerpiece, 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 currently selected information. When some data of a service is received at the *managed system*, it is notified using the appropriate interaction components for the current user situation and it is also showed in the master list. By selecting a notification of the list view, its details are shown. Fig. 6.12 shows these views for an implementation on iOS and Android. When the application is installed in the device, it registers the device to the infrastructure.

In the next subsection, we provide further detail of the implementation on the Android platform.

The manage 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[9] such as *Service*, which provides functionality executed in 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. The different components are generated automatically by means of model-to-text transformations using XPand templates as we described in Section 6.1.

In order to integrate our infrastructure with the manage system on Android, a *controller* was implemented as an Android Service running in background (see Fig. 6.13). This component was developed to act as a local *broker*. It supports the following operations:

Activation and deactivation of components. Android allows

[9] http://developer.android.com/guide/components/index.html: Android components

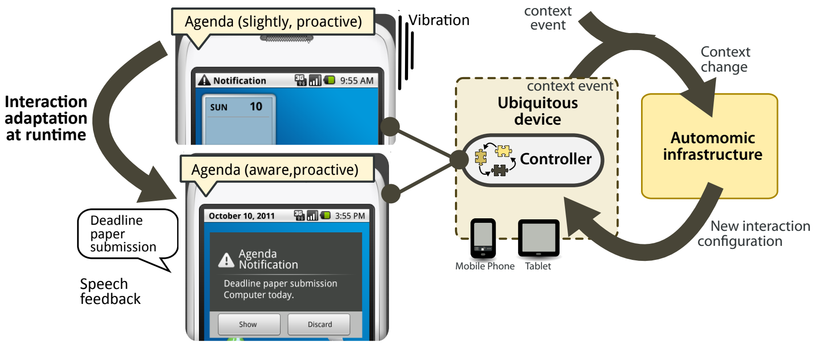


Figure 6.13: The components of the managed system on Android.

the dynamic activation and deactivation of components (services and activities). The different components executed in the mobile client are activated and deactivated by the controller as our infrastructure request it. 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). For locating a component in Android, an *Intent* is launched that describes the components to start. That is an abstract description of the component desired to be activated (e.g., vibration feedback) regardless of the component that provides such functionality (e.g., vibrator resource). For example, the code of Listing 6.15 launches a service that provides the "*es.upv.pros.Vibration*" interaction.

Listing 6.15: Excerpt of the implementation to launch a component on Android.

```
Intent i;
i = new Intent("es.upv.pros.Vibration");
startService(i);
```

An *Intent* can also carry small amounts of data to be used by the component that is started. For example, we can include in the vibration intent the pattern to be followed or the length of time to vibrate.

Dependency changes. Android provides loosely-coupled communication mechanisms among components. For requesting some external functionality, a component launches an *Intent*. In this case, the *controller* intercepts the signals from our infrastructure (abstract Intents) and translates them to a more specific ones that matches the concrete underlying technology. For instance, if the infrastructure indicates that a *notification* 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 later would be the component corresponding to the Android notification service in charge of showing status bar notifications (e.g., "*es.upv.pros.Notification.StatusBar*")

6.5 Applying AdaptIO to non-adaptive services

In this section, we discuss the steps necessary to apply AdaptIO to existing non-adaptive services. As a prerequisite, the service obtrusiveness adaptation behavior has to be modeled by means of the design method presented in Chapter 5. Then, the service interaction components are derived automatically from the model-based description. Once the different interaction components are prepared, the service has to be modified in order to be connected with the infrastructure by means of a service invocation. Specifically, the steps to be taken towards preparing adaptive services are the following:

1. *Identifying adaptable interaction components.* The initial step is to analyze the service and specify the required adaptability factors for each obtrusiveness requirement. After specifying the adaptability factors, it is necessary to identify the relevant supporting user interface components. During this process, a software en-

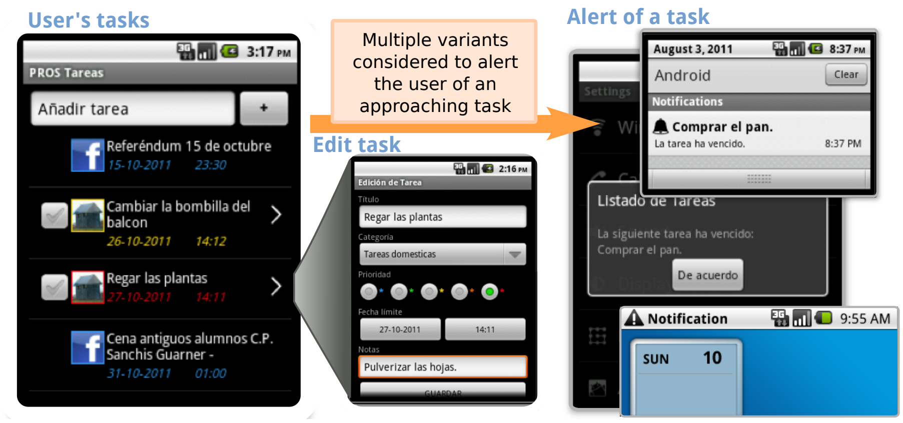
gineer analyzes the original software with regard to the list of potential adaptability factors to obtain the new requirements.

2. *Modeling the service adaptation behavior.* The service interaction obtrusiveness adaptation behavior has to be designed by following our modeling framework. First, the possible obtrusiveness levels must be defined and the interaction configurations for each level according to the adaptability factors. Also, if a new user situation is introduced, the rule to infer the user situation must be defined and inserted in the rules repository.

3. *Developing alternative components.* The next step is to transform or develop the source code of the alternative variants of the interaction components. Since we follow a model-based approach, the multiple variants can be obtained automatically following our development method. Our approach considers each interaction component as an isolated component that has its own functionality. However, if the degree of service functionality overlaps among different component variants, developers only need to provide the common functionality in a single point.

4. *Connecting the service with our infrastructure.* Finally, in order to integrate the service with our autonomic infrastructure, the interaction layer of the service has to be updated to take into account adaptation. That is, it has to invoke the *Service Manager* of our infrastructure via HTTP in order to provide to our infrastructure the information to show or communicate this information to the user adapted to the user's situation. However, these invocations can be reused between the different services.

In order to illustrate this process, we have modified a user's task management system to introduce the interaction obtrusiveness adaptation on it. This system allows users to add tasks and events under different categories with deadline information and priority. For the non-adaptive version of the system, the tasks were provided always using the same interaction components (a dialog). After applying our framework to the system, the interaction components available in the interaction


User's tasks
Multiple variants considered to alert the user of an approaching task
Alert of a task
Edit task

3:17 PM
PROS Tareas
Añadir tarea +
Referéndum 15 de octubre 15-10-2011 23:30
Cambiar la bombilla del balcon 26-10-2011 14:12
Regar las plantas 27-10-2011 14:11
Cena antiguos alumnos C.P. Sanchis Guarner - 31-10-2011 01:00

2:16 PM
Edición de Tarea
Titulo: Regar las plantas
Categoria: Tareas domesticas
Prioridad
Fecha limite: 27-10-2011 14:11
Notas: Pulverizar las hojas.
GUARDAR

August 3, 2011 8:37 PM
Android Clear
Notifications
Comprar el pan.
La tarea ha vencido. 8:37 PM

Listado de Tareas
La siguiente tarea ha vencido:
Comprar el pan.
De acuerdo

Notification 9:55 AM
SUN 10

Figure 6.14: Modifying a user's tasks management system to be adaptive.

layer for interacting with the system were: vibration, loud and soft audio, speech, an status notification, a dialog alert, and a home widget (see Fig. 6.14). These were the interaction component variants considered to adapt the system.

In order to connect this system with the autonomic infrastructure, we reused the controller component described in the previous section (Section 6.4) since it was also implemented on the Android platform. Specifically, we integrated this component in the interaction layer of the application in a way that, before calling an interface, the controller processes the interaction configuration from the autonomic infrastructure and makes the appropriate calls to the components.

6.6 Scalability evaluation

Model-based techniques provide a greater degree of flexibility for supporting the interaction obtrusiveness adaptation. By following an adaptation approach based on models, we can adapt the interaction in terms of concepts of a high level of abstraction. Since the models are directly interpreted, we use the same representation at runtime than at design time. Therefore, when the models change, the system is automatically updated.

However, model manipulation at runtime may impact overall system performance. Specifically, the incorporated latency is determined by (a) *the model manipulation frameworks*, (b) *the model population* (including the number of services taken into account), and (c) the *web service invocations*. Since the scalability of ubiquitous services is an important problem rarely addressed (Mostéfaoui et al., 2004), we have evaluated this concern in our autonomic infrastructure. As model manipulation frameworks, we used SPARQL to manage the context model and EMF Model Query to query the models at runtime. Specifically, we demonstrate the feasibility of exploiting at runtime the models introduced in our approach.

Experimental setup. The target platform 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.

In the adaptation of our running services (Smart Home services), model processing did not introduce significant performance penalization. However, in order to validate the efficiency of the proposed approach to large systems, we quantified this overhead for randomly generated models. These models started with one element and were populated with five hundred new elements each iteration. After the model population, we applied the model operations that support the calculation of the interaction modifications: *InsertContextEvent, checkUserSituation, triggerTransition, getDifferences, and doWeaving*. Our system requires these operations to be efficient enough to gather the necessary knowledge without drastically affecting the system response.

Figure 6.15 shows the milliseconds that take the infrastructure performing each one of the model operations. Analyzing in detail the experimentation results, we notice that the operations with the highest temporal cost were the operations related to trigger a transition (*trigger-*

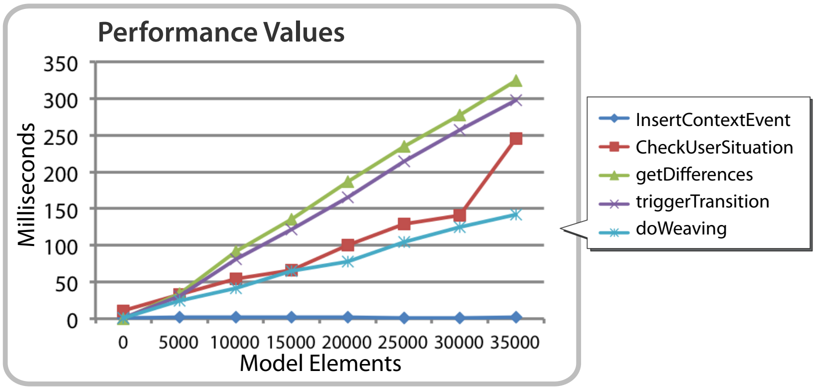


Figure 6.15: Values of infrastructure performance.

Transition and *getDifferences* operations). These operations navigate through the models to obtain the configurations and calculate the differences among them. For this reason, both operations get a similar time response. The operation over the context model that got higher response time was the *CheckUserSituation* operation, since it queries the ontology by means of SPARQL. Although the *doWeaving* operation also operates over the models, it got less response time because it only queries the elements involved to do the weaving (it does not navigate through all the model). Finally, the *InsertContextEvent* operation do not introduce any penalization in the response time. Overall, even with a model population of 35.000 elements in each model, the model operations provide fast response time (<350 milliseconds) at least for the kind of services we are addressing.

To determine the performance of a REST-based architecture in our approach, we evaluated the web service invocations. The evaluation was performed using the university wireless network, and the REST-based invocations were performed on a HTC Magic device. We performed 500 invocations in sequence, measuring the time from the creation of a request to the parsing of the result. Approximately 85% of the requests were shorter than 290 ms. The longest duration was close to 1 second (this could be due to a socket timeout on the device).

It turns out that our approach gathers the necessary knowledge from the runtime models without drastically affecting the system response. Also, we consider that the response time offered by our model operations is acceptable when compared to the performance of the devices and communication networks usually found in the Smart Home domain. Thus, we can conclude that our approach can also be applied in other domains with similar temporal constraints.

6.7 Conclusions

In this chapter, we have used model-based adaptation techniques to self-regulate service interactions in terms of obtrusiveness. With the increase in the capabilities of mobile and ubiquitous devices, user attention becomes a bottleneck for the system. Therefore, it is important to be able to manage it in an effective manner. Our infrastructure takes into account the user's attention for adapting the way in which service interactions are delivered 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 modeling level allows the system to be sustainable since it can support its evolution to new technologies.

7

Exploiting the User Feedback

Improving the obtrusiveness design

In any design that is going to last more than a very brief moment, the environment, the context, the expectations of people who use it, are bound to change.

—*Mark D. Gross.*

As the use of ubiquitous devices (e.g., mobile phones, tablets, TVs, etc.) is widespread nowadays, they increasingly ring, beep or show information at inappropriate situations or contexts. For example, when ringing in a meeting or in a theatre, or when showing inopportune information with company. Sometimes, we receive reminders to turn off our mobile devices or silent them, however, we often forget to do so resulting in unwanted interruptions. Even when we do remember, we then forget to turn on the device afterwards resulting in missed notifications, service information or calls (Rosenthal et al., 2011).

In Chapter 5 and Chapter 6, we presented a method and an infrastructure for designing and adapting services to regulate their interaction obtrusiveness at runtime according to user's current situation. However, users' context and circumstances can change over time and the defined service obtrusiveness behavior must be re-adjusted to adapt to these changes. Otherwise, the adaptation of these services' interactions not only may become useless for users but may also become a burden on them. Although the proposed models specify the interaction behavior of services according to user's context, changes in user behavior and preferences cannot be anticipated at design time. This highlights the need to adapt the decisions made at design time by learning the users' preferences through experience.

Analyzing user behavior (based on user's reaction or user's ongoing interaction with the system), the user personality can be better understood and interaction obtrusiveness can be improved. Every time the system suggests a choice to the user, he or she accepts it or rejects it, giving feedback to the system to update its knowledge base (e.g., models) either *implicitly* or *explicitly* (Jaimes and Sebe, 2007). In this way, interaction obtrusiveness adaptation can be conducted implicitly by the system or explicitly by the end-user. User history can be monitored implicitly, by observing the user interacting with the system. Additionally, the user can explicitly characterize his/her own preferences and modify the behavior of the adaptation by means of tools supporting customization (Toninelli et al., 2008).

In this chapter, we introduce a **learning strategy** for adapting the designed service interaction obtrusiveness implicitly (specified in the unobtrusive adaptation space) based on user's reaction. Specifically, we follow a reinforcement learning approach for automatically adapting the interaction obtrusiveness in a way that maximizes the user's satisfaction for long-term use. The initial interaction obtrusiveness design ensures us a consistent initial behavior according to initial user needs. This design is then adapted for each service to the individual behavior and preferences of users based on the feedback given by them.

Also, we provide users with **customization interfaces** in order to let them explicitly set up their own obtrusiveness preferences, since the user is who better knows his/her new preferences. In particular, we have developed a mobile interface with expressive user support for defining situations and a personalization interface to allow users to change the services in the unobtrusive adaptation space and the linking with the interaction resources.

Because this interaction obtrusiveness must learn from observing the behavior of users, another distinguishing characteristic of these systems is their need for **rapid learning** (Jaimes and Sebe, 2007). An important issue is the number of training cases needed by the system to generate good adaptation. Thus, we have conducted an experiment to evaluate the accuracy of our learning method.

The rest of the chapter is organized as follows. We first motivate the problem and characterize the learning approach in Section 7.1. Then, we describe the accomplished method to adapt automatically the initial design based on user feedback in Section 7.2. Next, Section 7.3 describes the provided interfaces to change the design explicitly. Afterwards, we present experimental results in Section 7.4. Finally, Section 7.5 concludes the chapter.

7.1 Characterization of the obtrusiveness adaptation

According to the unobtrusive adaptation space presented in Chapter 5, there is a range of ways to provide the interaction of a service with different degrees of attentional demand and proactivity. Which is the best? It depends on the *user* who is interacting with the system (e.g., his/her needs, priorities, preferences, context), the *service* (e.g., allowed obtrusiveness levels to be presented), and the *context* (e.g., environment, location, user state, cognitive load, etc.). We take into account all these factors to define the initial service obtrusiveness design.

But, what happens if user preferences change over time? How the user reacts to a designed interaction? Which is the new user behavior? What happens if a new service appears in the system?. There are a lot of questions that raise the need to adapt the initial obtrusiveness design to the new user preferences in order to ensure user satisfaction with the system over time.

The different factors that could affect the initial design according to our characterization and the way to adjust it are the following.

- **User profile:** User profile can change over time due to a change in the user's lifestyle or user's needs. In this situation, the user could need an adaptation of the type of user to another profile. Thus, services and obtrusiveness configurations should be adapted to this new profile.
- **User preferences:** A specific preference within the same type of user could change and the user could need another obtrusiveness level for a specific service in a context. The system should be able of adjusting the interaction obtrusiveness according to the new user preferences.
- **User context:** User's context can change and the user situation definitions of the transitions between two obtrusiveness levels could require a re-definition in order to guarantee accuracy in the interaction adaptations. Thus, the system should offer a manner to change the definition of the user situations and, in this way, re-define the transitions.
- **Services:** Users can need new services or existing services can change their functionality over time. If a new service appears, the system should be able of learning the relevant obtrusiveness levels for the service. If an existing service changes its functionality, the user may prefer another obtrusiveness level for the new functionality. Thus, the system should adjust the obtrusiveness level for that service.

From this characterization, we propose three types of adaptations:

- **Type 1**: adaptations between user profiles.
- **Type 2**: adaptations of obtrusiveness levels for a service.
- **Type 3**: adaptations of user situations of transitions for a service.

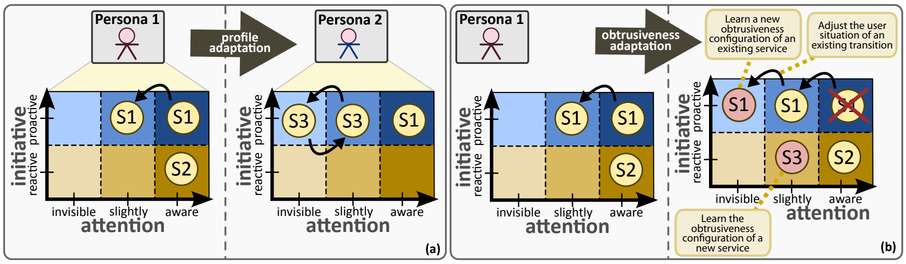

Persona 1
profile adaptation
Persona 2
initiative
reactive proactive
attention
invisible slightly aware
S1
S2
S3
(a)
obtrusiveness adaptation
Learn a new obtrusiveness configuration of an existing service
Adjust the user situation of an existing transition
Learn the obtrusiveness configuration of a new service
(b)

Figure 7.1: Types of adaptations: (a) profile adaptations, (b) obtrusiveness levels and transitions adaptations.

For example, an unemployed user can have a specific profile that could need to be adapted if the user begins to work (from *unemployed profile* to *worker profile*). In addition, a particular user can play two roles at different moment (e.g., weekdays vs. weekends) and the system could adapt the profile according to it (adaptations of type 1). Moreover, a specific service within a profile could need to be adjusted. Imagine it was defined for a user the *healthcare* service to be always in the *aware* level of attention by means of *speech* feedback. However, when the user begins to work s/he does not want other people to be aware of it. Thus, this obtrusiveness level for the service needs to be adjusted to another one that requires less attention (adaptation of type 2). Also, the user's context can change in a way that s/he has changed his/her job and s/he had specified a user situation rule *atWork* with the location of his/her old job in its definition. However, as the job location has changed, this rule should be adjusted with the new job location (adaptation of type 3).

Figure 7.1 illustrates these types of adaptations. In the first case (see Fig 7.1(a)), profile adaptation can be defined by designers in the design phase by means of a transition between *Personas*. The way to define transitions between personas is similar as the one to define transitions among obtrusiveness levels defined in Chapter 5. However, in this case, the *action* of the transition is the target persona. In the second case (Fig 7.1(b)), the system has to improve the initial design by adapting the obtrusiveness level of a service. Our system achieves this by learning from user's reaction after interacting with the system in a specific obtrusiveness level. This case of adaptation is described in Section 7.2. The users can also set up their new preferences explicitly by means of an end-user interface described in Section 7.3. In the third case (see Fig. 7.1(b)), the user is put in charge of defining his/her own situations by means of a mobile interface described in Section 7.3. In the case that a new service appeared in the system, it is located in a default obtrusiveness level (e.g., slightly-aware obtrusiveness level) and this obtrusiveness level is adjusted in the same way of an existing service by means of an adaptation of type 2 (second case described).

The following sections provide further details about the mechanisms that we propose to address each type of adaptations.

7.2 The reinforcement learning strategy

As it is difficult to model interaction obtrusiveness adaptation in the design stage to meet the changing users' preferences over time, we need mechanisms to learn these user preferences automatically (Maes, 1994). Thus, we propose to **automatically learn the obtrusiveness preferences and adapt the initial design at runtime**. Our system achieves this by learning from user's reaction after interacting with a service in a specific obtrusiveness level.

In order to deal with user preferences, some research try to inject user models into context-aware systems. For example, (Byun and Cheverst, 2001b) showed how the two closely related concepts of user modeling and context-awareness could be combined. (Godoy and Amandi, 2005) built user profiles of net surfers capturing their preferences and interests for page suggestions. These profiles are learned using classification techniques. (Doctor et al., 2005) used fuzzy logic to learn rules representing the user's behavior with the aim of automating his/her usual actions. Following this direction, our goal is to learn the preferences of the user based on the satisfaction or disapproval of the user towards the system's actions.

In order to improve the initial obtrusiveness design and adapt the obtrusiveness level for each service, we follow a *Reinforcement Learning* (RL) strategy (Sutton and Barto, 1998b) in which our system learns from user feedback. Reinforcement learning is an approach where an agent acts in an environment and learns from its previous experiences to maximize the sum of rewards received from its action selection. RL has been demonstrated to be a promising approach in autonomous systems to avoid knowledge bottlenecks by automatically learning high-quality management policies (Tesauro, 2007).

In the next subsections, we explain how we have applied RL in our approach to adapt the initial obtrusiveness knowledge.

7.2.1 Applying RL to our approach

Considering the obtrusiveness levels as the different states of our system and depending on the received feedback (positive or negative), the current obtrusiveness level for a service is rewarded or punished. In this way, if negative feedback is received continuously for a given service in an obtrusiveness level, it means that the user is unsatisfied with the design and the system will adjust it by changing the obtrusiveness level. Thus, in each obtrusiveness level, the possible actions that the system can take are:

- *do nothing*: for maintaining a service in the same obtrusiveness level while positive feedback is received.
- *adjust the obtrusiveness level*: for improving the designed obtrusiveness level for a service due to received negative feedback.

To select an action in each obtrusiveness level, we define a *behavior policy* with the goal of optimizing all subsequent feedbacks over time. Each time feedback is received, the weights of each action are updated based on the new information. The process is the following: while positive feedback is received, the action *do nothing* increases its weight and the policy will choose this action since it is defined to choose the action

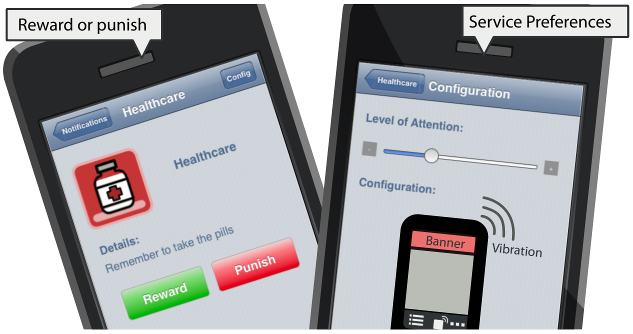


Figure 7.2: Ways of obtain explicit feedback from user.

with the highest value in a given state (obtrusiveness level). However, if the system starts to receive negative feedback, *do nothing* action will start to decrease its weight until the action to *adjust the obtrusiveness level* has a higher value (according to a learning rate). When this situation happens, the system will proceed to adjust the obtrusiveness level since the system will choose the action to *adjust the obtrusiveness level* and will adapt the design. The learning rate determines to what extent the new value feedback overrides the old one. It is initially defined manually and then can be adjusted to each user.

7.2.2 Obtaining the Feedback

Since the user is the target of the system, the user is who can reward or punish it by giving feedback (a reward r). We specify a reward of $+1$ to express satisfaction and -1 to express disapproval. This feedback can either be explicit or implicit. *Explicit* feedback is obtained directly from the user by means of two verbs that appear in the service interface as implemented in (Isbell et al., 2001): *reward* and *punish*. Also, if users change their service preferences, that is considered explicit feedback. Fig. 7.2 illustrates the ways to obtain explicit feedback.

Implicit feedback is obtained by observing user's behavior (interaction with the system) after interacting with a service in a specific obtrusiveness level. Specifically, it is detected as the immediate interaction related to the service at hand that is performed by the user. For example, if the user receives information from a service and s/he views it, then s/he probably was satisfied with the action. However, if the user changes the feedback mode when s/he receives it, then the implicit reward is negative. This semantic is specified by means of a function that takes the immediate interaction as input and returns the adequate value (positive or negative).

For a service S_i delivered to the user, we define the valued feedback as $r_i = \alpha R_E(i) + (1-\alpha)R_I(i)$, where $R_E(i)$ is an explicit feedback, $R_I(i)$ is an implicit one, and α is a regulating factor that adjusts the ratio of implicit and explicit feedback. In our approach, implicit feedback is numerically weaker that the explicit one.

7.2.3 Running the Reinforcement Learning algorithm

Through an online trial-and-error process by means of actions, our system is able to learn a decision-making policy which optimizes all subsequent rewards over time. Also, using an *a-priori* designed model, we avoid to explore particular actions that are heavily punished but rather reason about the consequences of these actions (Wiering, 2002). For example, with the a-priori model of our system, the agent can compute a policy to attain its current goal (maintain the obtrusiveness level). If the agent discovers more information about the user, it can again instantiate this in the model and recompute a policy without the need to try all its actions.

Specifically, the RL method we consider to update the weights is called *Q-learning* (Watkins and Dayan, 1992). This method has a function Q that calculates the *quality* of choosing an action a in the obtrusiveness state s (state-action combination $Q(s, a)$). Before learning has started, Q returns a fixed value chosen by the designer. This value is specified following our initial design by giving a higher value to the pairs corresponding to the default obtrusiveness level and the “do nothing” action, and a less value to the remainder. Then, each time the user gives a reward, new values are calculated for these pairs as follows:

$$Q(s,a) \Leftarrow Q(s,a) + \alpha(r + \gamma max_{a' \in A(s')} Q(s',a') - Q(s,a)) \tag{7.1}$$

Here, $Q(s,a)$ is the old value of the pair state-action; α is the learning rate that determines to what extent the new feedback overrides the old one. A factor of 0 makes the system not learn anything, while a factor of 1 would make it consider only the most recent feedback; r is the reward observed after performing the action a in an obtrusiveness state s; γ is a discount parameter between 0 and 1 expressing the importance of future rewards; and $Q(s',a')$ is the expected future feedback. We have introduced a new function V to establish a limit value that the quality value can take to accelerate the learning and obtain an adaptation with few trials when the user changes his/her preferences. This new function is:

$$V(s,a) = \left\{ \begin{array}{cl} -\Delta & if\ Q(s,a) <= -\Delta \\ Q(s,a) & if\ -\Delta < Q(s,a) < \Delta \\ \Delta & if\ Q(s,a) >= \Delta \end{array} \right. \tag{7.2}$$

Where Δ is the limit value that the quality value can take. This value has to be defined according to the reward and learning rate values.

However, in our system, even though when the action to choose is "do nothing" while receiving positive feedback, the obtrusiveness level (state) can be modified by a change in the user's situation. In this case, we assume the action "do nothing" and the new obtrusiveness level as the new state. Algorithm 1 shows our modified version of *Q-Learning* for learning after interacting with a service and taking into account the limit value function and the changes in the user situation.

As we have a designed a-priori model, we initialize s as the designed default obtrusiveness level and the value of the state-action pairs $Q(s,a)$ and $V(s,a)$ with the designed values giving more weight to the action *do nothing* in the initial obtrusiveness state. This provides a consistent initial behavior. When a service has to interact with the user, the system choose an action a in the state s following the behavior policy.

Algorithm 1 RL algorithm for a service

```
Require: discount factor $\gamma$, learning parameter $\alpha$, limit value $\Delta$
 1: Initialize $s$ with initial obtrusiveness state, and $Q(s,a) = V(s,a)$
    with initial values
 2: repeat
 3:     when service $S_t$ has to interact with the user
 4:     choose action $a$ in $s$ using behavior policy
 5:     perform $a$
 6:     perform service interaction in the new state $s'$
 7:     observe received reward $r$
 8:     update $V(s,a)$
 9:     $s := s'$
10:     if obtrusiveness change by user's situation from $s$ to $s''$ then
11:         $s := s''$
12:         $a :=$ "do nothing"
13:     end if
14:     if r is negative then
15:         save the kind of negative reward
16:     end if
17: until end of learning process
```

Then, the action is performed and the service interaction is performed in the new obtrusiveness state s'. The system observes the reward obtained from the user either explicitly by means of a graphical interface or implicitly by recognizing the user's emotional state. Based on the obtained reward, the system updates the value of the state-action pair. Then, s is updated to the current obtrusiveness state s'. The system also checks if the obtrusiveness state has been modified due to a change in the user's situation. If so, the s and a values are updated with the new obtrusiveness state s'' and the "do nothing" action. Finally, if the received reward is negative, we save the kind of negative reward received.

7.2.4 Adjusting an obtrusiveness level

If the action to perform is to adjust the obtrusiveness level for a service in a given user situation, the system has to compute the *initiative* and *attention* level required for the adaptation to a new obtrusiveness level. We introduce an ordering in the obtrusiveness axes as $invisible < aware$ ($aware$ interactions requires more attention than $invisible$ ones) and $reactive < proactive$ (a *reactive* value provides a lower degree of automation than the *proactive* one). In this way, we express the changes in the obtrusiveness level as increments and decrements in the different axes.

To calculate the increments and decrements, we process the kind of negative feedback. For example, if the user puts the device into silent mode after receiving a notification, it means a decrement in the attention axis (decremental feedback). However, if the user is not aware of a notification, it needs to increment the attention (incremental feedback). For processing this feedback, we calculate the median for all the increments and decrements as explained in Section 5.3. The rationale behind it is to obtain the average movement in the unobtrusive adaptation space. Finally, the resulting action is applied.

7.2.5 An application example

In order to exemplify the application of the algorithm and the way the reinforcement strategy works, we based on an example of the *healthcare* service. Initially, this service was designed in the *(aware, proactive)* obtrusiveness level being this state able to be changed to the other proactive levels due to different user situations. Figure 7.3 shows the unobtrusive adaptation space of the service with its transitions.

Imagine a period of time in which the *healthcare* service notifies the user about taking the pills when s/he is having lunch alone. According to the initial obtrusiveness design, the interaction is performed in the *(aware, proactive)* level of attention (e.g., be means of speech feedback). Suddenly, the user changes his/her routine and starts to having lunch with friends. Following the obtrusiveness design, the service carries on in the *(aware, proactive)* obtrusiveness state. However, in this new situation, the user starts to give negative feedback because s/he switches

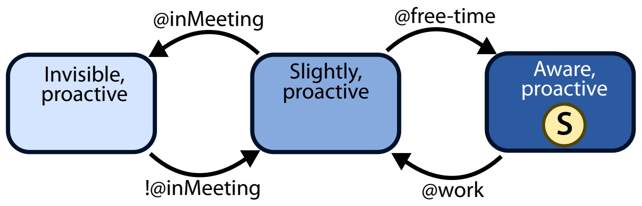
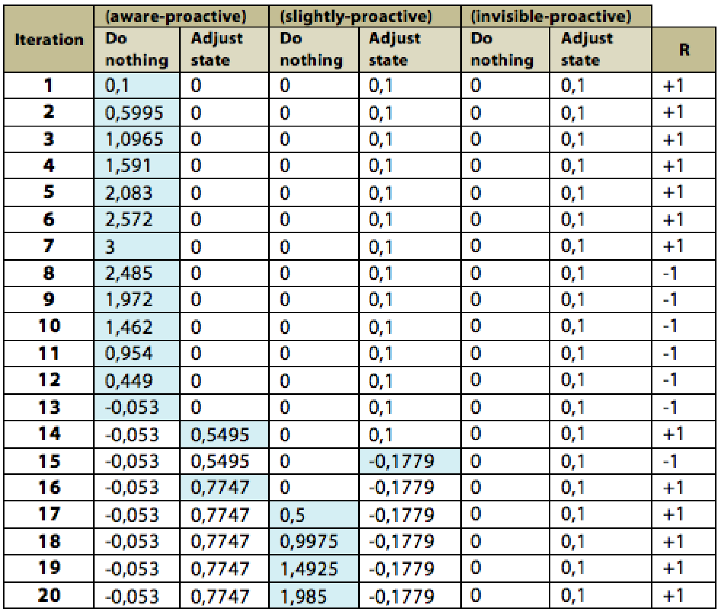


Figure 7.3: Unobtrusive adaptation space for the healthcare service.



Iteration	(aware-proactive)		(slightly-proactive)		(invisible-proactive)		R
	Do nothing	Adjust state	Do nothing	Adjust state	Do nothing	Adjust state	
1	0,1	0	0	0,1	0	0,1	+1
2	0,5995	0	0	0,1	0	0,1	+1
3	1,0965	0	0	0,1	0	0,1	+1
4	1,591	0	0	0,1	0	0,1	+1
5	2,083	0	0	0,1	0	0,1	+1
6	2,572	0	0	0,1	0	0,1	+1
7	3	0	0	0,1	0	0,1	+1
8	2,485	0	0	0,1	0	0,1	-1
9	1,972	0	0	0,1	0	0,1	-1
10	1,462	0	0	0,1	0	0,1	-1
11	0,954	0	0	0,1	0	0,1	-1
12	0,449	0	0	0,1	0	0,1	-1
13	-0,053	0	0	0,1	0	0,1	-1
14	-0,053	0,5495	0	0,1	0	0,1	+1
15	-0,053	0,5495	0	-0,1779	0	0,1	-1
16	-0,053	0,7747	0	-0,1779	0	0,1	+1
17	-0,053	0,7747	0,5	-0,1779	0	0,1	+1
18	-0,053	0,7747	0,9975	-0,1779	0	0,1	+1
19	-0,053	0,7747	1,4925	-0,1779	0	0,1	+1
20	-0,053	0,7747	1,985	-0,1779	0	0,1	+1

Figure 7.4: Quality values for 20 iterations of the algorithm.

off the volume of his/her mobile device every time s/he receives the notification while having lunch. This means that the user prefers an obtrusiveness level with a lower attention level (decremental feedback). Thus, in this situation, our system learns this new preference by means of the reinforcement learning algorithm.

Figure 7.4 shows the trace of the algorithm for 20 iterations. The

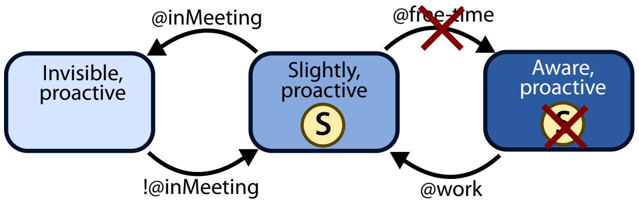


Figure 7.5: Updated unobtrusive adaptation space for the healthcare service.

parameters were initialized in the following manner: γ=0'99, α=0'5, and Δ=3. Cells in blue shows the updates of the quality values each time feedback was received. Initially, the values were defined based on the initial obtrusiveness design. In this particular example, we gave priority to the action "do nothing" in the *(aware, proactive)* obtrusiveness state. While positive feedback is received in this state, the quality value for the "do nothing" action in that state is increasing after the limit value (see iteration 7). In the iteration 8 the user changes his/her preferences wanting less attention (see that the reward is negative in this iteration). Thus, the system begins to decrease the quality value of the current state due to the received negative feedback until it starts to learn the new obtrusiveness state (iteration 14 to 16). Finally, the system stays stable in the learned obtrusiveness level due to received positive feedback (from iteration 16).

When the obtrusiveness level for a service is adjusted according to the new user preferences, the *unobtrusive adaptation space* is updated to reflect the changes. For example, according to the learned preference in the example of the healthcare service, this service cannot transit anymore to the *(aware, proactive)* obtrusiveness level. Thus, the transition from the *(slightly, proactive)* level to the *(aware, proactive)* level is deleted as illustrated in Fig. 7.5.

Note that this update of the model is produced after the period of learning when the obtrusiveness level becomes stabilized. In the example, this update is performed in the iteration 19. In this way, we avoid unnecessary updates while the system is learning.

7.3 The customization interfaces for end-users

Another possibility to learn user preferences over time is that users characterize their own preferences explicitly (Toninelli et al., 2008). To contend with this, we need a tool that lets end-users easily and efficiently customize their interaction obtrusiveness. To that end, we have developed mobile interfaces to allow users adjust and customize the initial obtrusiveness design.

End-user centered approaches provide alternatives for end-users to define the behavior of their systems and environments. Dey et al. (Dey et al., 2004) presented a programming-by-demonstration approach for prototyping context-aware applications. They experimented with a prototype PC-based tool that lets users train and label context models, which can then be mapped to actions. Chin et al. (Chin et al., 2006) provided a platform aimed at non-technical people to customize the functionality of their digital home to suit their particular needs. Korpipää et al. (Korpipaa et al., 2006) provided users with a UI to manually define new interactions in smart phones (e.g., gestures) and link them to device actions. Usually, these techniques improves user acceptance of the system because they provide users with better control over the system. However, these techniques do not focus on specifying the degree of obtrusiveness required to provide the functionality. In contrast, our designed interfaces allow users to define fine-grained obtrusiveness adaptations.

Because mobile devices are lighter and more portable, we find it more convenient to develop the customization interfaces for them. In this way, **users can customize their system at any time and everywhere**. However, unlike traditional desktop interfaces, mobile interfaces have several constraints such as limited screen, mobility, viewing conditions, etc. and it is very important from the designers' viewpoint that the development of this kind of interfaces follows a user-centered design process to arrive at the end-users[1].

Furthermore, interaction aspects are a crucial factor for determining the application's success or failure since a bad interaction can cause that user does not use the application. Specially, we have put emphasis in **simplicity** as one of the main goals of our design solutions. Simplicity is about subtracting the obvious and adding the meaningful (Maeda, 2006). This principle is behind the design of successful products such as the iPod (Maeda, 2006).

Many usability guidelines try to aid users in fulfilling their goals effectively, efficiently and successfully in each context of use (Iso, 2010). In order to obtain simpler designs, designers normally apply heuristic analysis to their interaction designs. There are different works describing heuristics or guidelines for general purpose design such as (Unger and Chandler, 2009), (Krug, 2005). There are also references related to touch gestures such as (Villamor et al., 2010), for voice user interface design (Cohen et al., 2004), or for mobile design (Savio and Braiterman, 2007). In many cases, simplicity is behind some of these guidelines. Nevertheless, by considering simplicity as an independent aspect for the system, some of the limits that reduce the design space are eliminated (Vastenburg et al., 2008).

Since we wanted to obtain simple interfaces in an early step, we inspired in the "*Laws of Simplicity*" of Maeda (Maeda, 2006) in order to design our customization interfaces. These guidelines are metaphors which are applicable to different areas such as design, business, technology and life. In our work, we took some of these guidelines to detect and fix complexity problems. Specifically, in our work, we apply the following laws:

- **Reduce.** To preserve the most relevant information and operations in the main screen, diverting the rest to other contexts where user can navigate.
- **Organize.** To group similar components and operations.

[1] http://www.uxbooth.com/articles/designing-for-mobile-part-1-information-architecture/

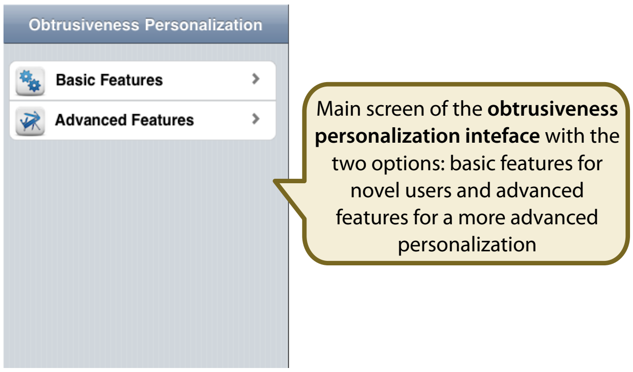

Obtrusiveness Personalization
Basic Features
Advanced Features
Main screen of the **obtrusiveness personalization inteface** with the two options: basic features for novel users and advanced features for a more advanced personalization

Figure 7.6: Main screen of the *Obtrusiveness Personalization* interface.

- **Time.** To avoid unnecessary navigation with the purpose of users complete their activities more fluently.
- **Learn.** To apply popular and intuitive knowledge that is movable to an application.
- **Emotion.** To achieve users have good feelings which lead to a satisfying interaction.

According to these laws, we defined our customization interfaces in different iterations in which new ideas were emerged. At the end, we obtained interaction designs more focused in the end-user. In the following subsections we describe the design of each one of the customization interfaces provided based on simplicity.

7.3.1 Obtrusiveness Personalization interface

To overcome the problem of the users' changing preferences and allow them to define their own obtrusiveness preferences, we provide an *Obtrusiveness Personalization* interface. This corresponds to the adaptation of *type 2*.

In particular, we developed a personalization interface to let users change the services in the unobtrusive adaptation space and the linking with the interaction resources in a user-friendly interface. The interface is based on the nested doll pattern (applying the "*reduce*" and "*time*" laws). Figure 7.6 shows the main screen of the interface where we show the main operations (applying the "*reduce*" and "*organize*" law). Below, we first discuss how the user can change the obtrusiveness level of a service on a basic interface, and then, how he can change the interaction resources assigned to each obtrusiveness level and the transitions among the levels in a more advanced interface.

Changing the obtrusiveness level (basic features)

We provide a basic interface for novel users to change the obtrusiveness level of services following the "*emotion*" law. As Figure 7.7 shows, when the user selects the basic features option, a list of all services appears (left screen). Selecting a service, its obtrusiveness levels with their associated user situations are shown. In this interface, when the user selects a specific user situation of the selected service, the obtrusiveness level associated with that user situation is shown (represented as the level of attention in the interface). In this screen, the user can choose one obtrusiveness level to change it (see Fig. 7.7, right screen) or delete it.

As the figure shows, the interface to change the obtrusiveness level is designed with a slider to let users increase or decrease the attention level without any knowledge about obtrusiveness levels (following the "*learn*" law). When the attention level changes, the configuration image changes accordingly to show users the interaction resources of the selected level (applying the "emotion" law).

Changing interaction resources and transitions (advanced features)

For advanced users, we provide an advanced interface to allow users (1) change the interaction resources assigned to an obtrusiveness level, (2) create, edit or delete transitions between obtrusiveness levels, and (3) manage conditions (user situations). Figure 7.8 shows the main screen of the advanced features (following the "*reduce*" and "*organize*" laws).

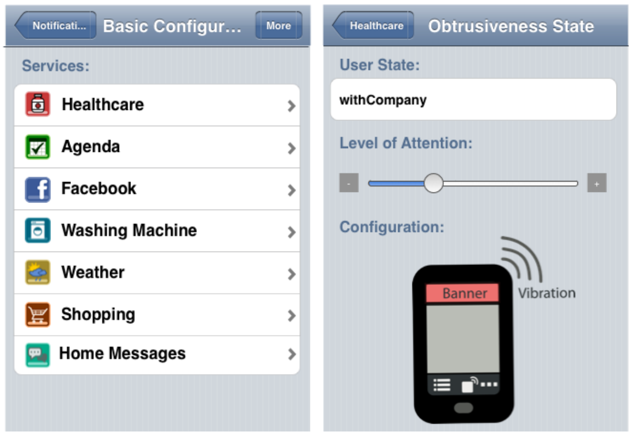


Figure 7.7: Interfaces for changing the obtrusiveness level.

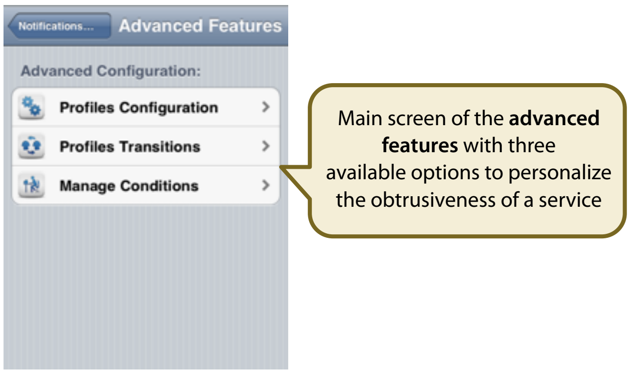

Notifications...
Advanced Features
Advanced Configuration:
Profiles Configuration
Profiles Transitions
Manage Conditions
Main screen of the **advanced features** with three available options to personalize the obtrusiveness of a service

Figure 7.8: Main screen of the advanced features.

Selecting one of the items lets users to change each one of the features.

Profiles Configuration. To change the interaction resources of an obtrusiveness level, the user can select an obtrusiveness level between the existing ones. This is done by selecting the row where the obtrusiveness level (obtrusiveness state) is shown (see Fig.

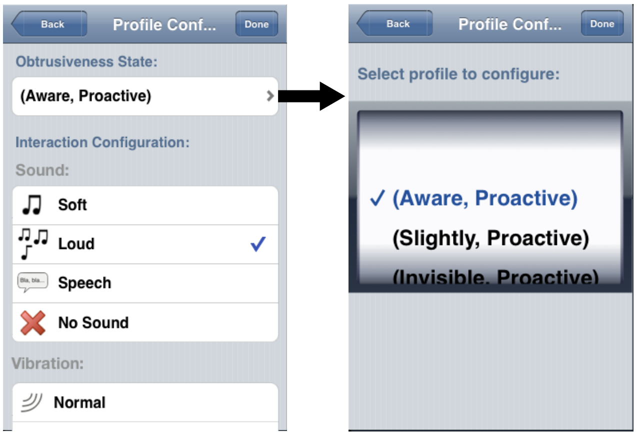


Figure 7.9: Profiles configuration screens.

7.9, left screen). In order to choose an obtrusiveness level, a list of all the obtrusiveness levels is shown to the user by means of a *picker* (see Fig. 7.9, right screen). When the user selects one, the profile configuration interface shows the complete list of interaction resources with the current configuration highlighted (with a "tick"). In this screen, the user can select *edit* to change this configuration by clicking other interaction resources. It is worth noticing that the selection follows the constraints defined in the interaction model.

This interface has been designed following the "*organize*", "*learn*", and "*emotion*" laws since the interaction components are organized in categories, an image of each interaction component is shown to provide more intuitiveness, and users can change the configuration with few interactions.

Profiles Transitions. To create, edit or delete transitions between obtrusiveness levels, a list of all the transitions is shown to the user (see Fig. 7.10, left screen). Each row of this list shows the origin and the target obtrusiveness levels, and the services that

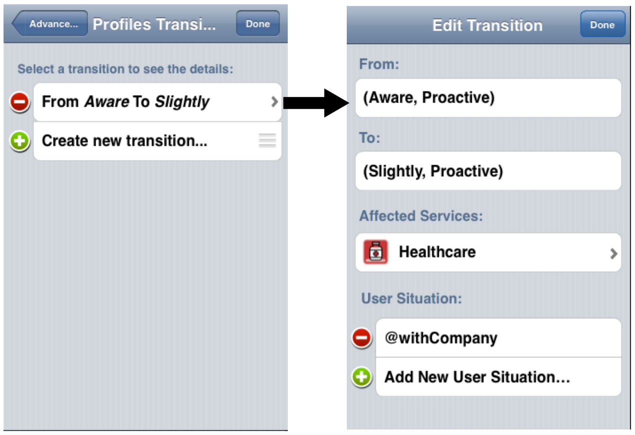
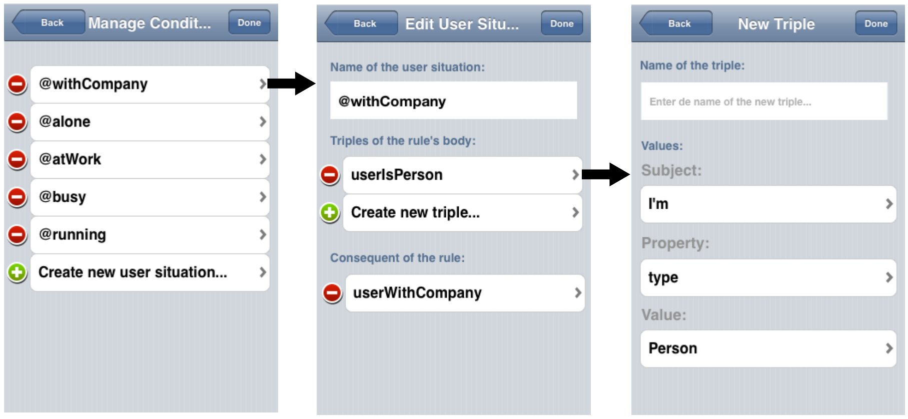


Figure 7.10: Profile transitions screens.

have the transition. In this screen, the user can select an existing transition from the list to edit its information or can delete it. The user can also create a new one by pressing the item "*Create new transition*".

When the user selects a transition to edit its information, the user has to define the *from* and *to* obtrusiveness levels of the transition (see Fig. 7.10, right screen). Then, he can choose the affected services in which the transition will be applied, and the user situations that make the obtrusiveness level to change. From this screen, the user can also go to the manage conditions options detailed below. This interfaces have been designed following the "*reduce*", "*organize*", and "*learn*" laws since the task has been divided in different screens, the interfaces have been organized by operations, and the used labels have been chosen to be intuitive for the users.

Manage Conditions. For creating, editing or deleting conditions (user situations), a list of all the user situations is shown to the user (see Fig. 7.11, left screen). From this list, the user can select



Name of the user situation:
@withCompany
Triples of the rule's body:
userIsPerson
Consequent of the rule:
userWithCompany

Name of the triple:
Enter de name of the new triple...
Values:
Subject:
I'm
Property:
type
Value:
Person

Figure 7.11: Manage conditions screens.

an existing one to edit its information or delete it. The user can also create a new one by selecting the "*Create new user situation*" option.

In the screen to create or edit a user situation (see Fig. 7.11, middle screen), the user can give a name to the user situation and add as many triples as s/he needs to define the body of the rule. At the end, the user can specify the consequent of the rule as a triple too. Each one of the triples is formed by the object (e.g., "*I'm*"), the property (e.g., "*type*"), and the value (e.g., "*Person*") as it is shown in the right screen of Fig. 7.11. The values to be chosen to form the triples are retrieved from the ontology context model. These interfaces follows the "*reduce*" and "*organize*" laws since only the relevant information is shown in each interface and this information is grouped and organized by sections.

7.3.2 User Situation Specification interface

In order to adapt the user situation definitions and guarantee accurate and unambiguous situation definitions, the user is put in charge of re-

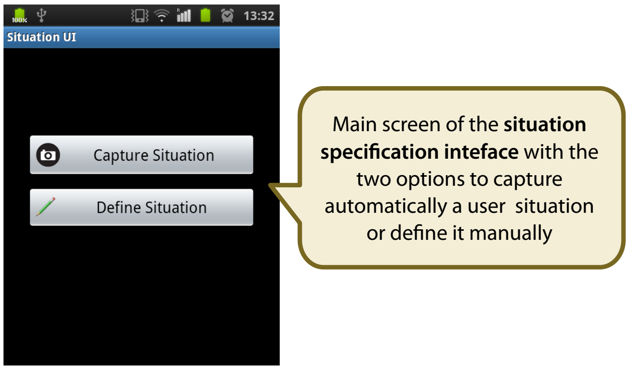

13:32
Situation UI
Capture Situation
Define Situation
Main screen of the **situation specification inteface** with the two options to capture automatically a user situation or define it manually

Figure 7.12: Main screen of the User Situation Specification interface.

defining his own situations. This corresponds to the adaptation of *type 3*.

We developed a mobile interface that allows users to specify their situations in a generic and fine-grained way, based on environment context. To increase usability and support nomadic users in a wide range of environments, the interface also supports directly capturing user situations (to support the "*time*" and "*learn*" laws). Figure 7.12 shows the main screen of the interface where we show the main operations (applying the "reduce" law). Below, we first discuss how the user can manually specify situations, and then, how s/he can use the "capture" functionality.

Manually defining situations

When the user chooses the "Define Situation" option, s/he can define a new user situation or re-define an existing one manually. In the first screen (see left screen of Fig. 7.13), the user can choose to define a still undefined situation (referenced in the obtrusiveness model), or edit an already defined one. In the second screen (see right screen of Fig. 7.13), s/he can define the chosen situation using two aspects: location and time. A third, more advanced "free-form" option allows the user to

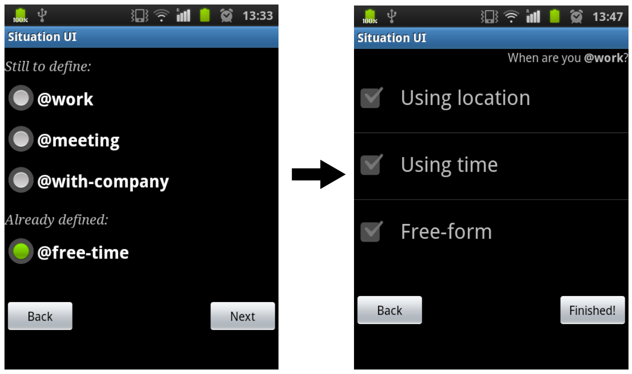


Figure 7.13: Interfaces of the situation overview and specification options.

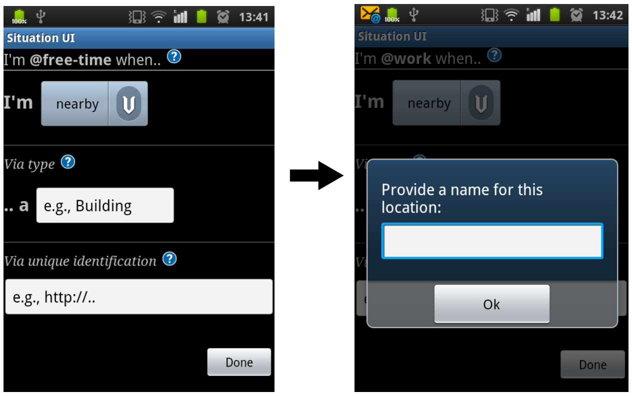


Figure 7.14: Interfaces to define a location.

place arbitrary constraints on his/her environment (see below). These interfaces have been designed following the "*organize*" law by grouping and organizing the functionality of them.

Using the location option, the user describes the location(s) s/he is in while being in the chosen situation. For each location (see Fig. 7.14), the user specifies whether s/he is inside or nearby a certain place, person or thing (i.e. physical entity) in that situation, and provides a way to identify that physical entity via its type and/or unique identifica-

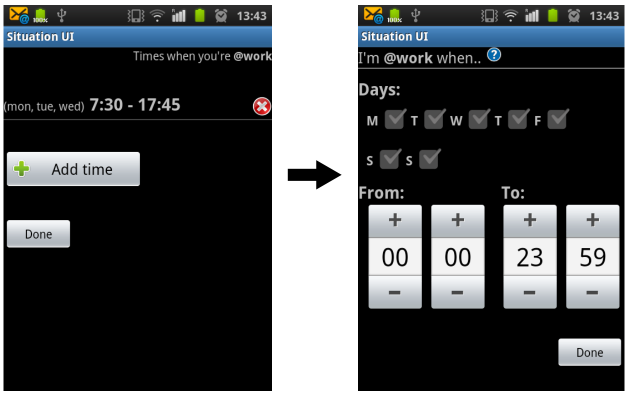


Figure 7.15: Interfaces to define time intervals.

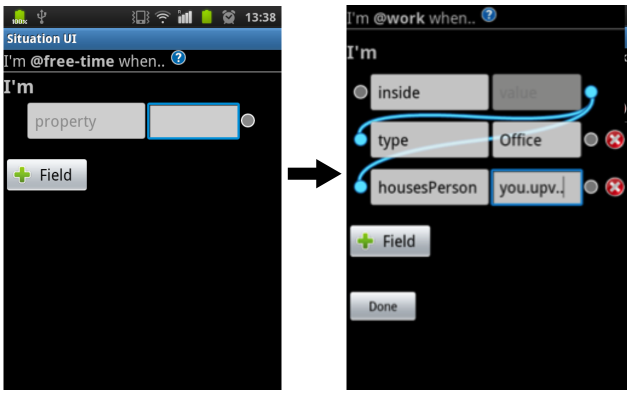


Figure 7.16: Interfaces to define the free-form option.

tion (URI). The user is aided via auto-complete functions (applying the "*time*" and "*learn*" laws): the type field suggests terms from the ontology, as well as synonyms of the ontology terms (provided by WordNet); while the URI field suggests URIs that identify physical entities the user has encountered.

The user can also specify time intervals (i.e., days of the week and time span) during which s/he is in the situation (see Fig. 7.15).

The advanced, “free-form” option (see Fig. 7.16) allows defining situations in a more powerful and expressive way, by placing arbitrary constraints on the user’s environment context. A constraint consists of a property and a value field. A user may arbitrarily constrain a property value, either by providing a concrete string or by linking its connector to other constraints. This free-form option, with connectable components, resembles the popular Yahoo! Pipes online mashup tool. In this example, the user is inside his office during the *@work* situation. To describe this in a generic way, the user specifies the *inside* property, and creates two constraints on the place s/he should be inside of. The first constraint states that the type of the place should be “*Office*”, while the second specifies that the place is the user’s office (via the *housesPerson* property). Using the constraint’s connectors, the user connects the two new constraints to the first constraint’s value field. The property and value fields are respectively backed by the same auto-complete functions mentioned above.

Capturing situations

The “capture” option exploits the user’s current environment to quickly and easily specify situations (to support the “*emotion*” law). In this option, the user takes a snapshot of his/her environment, fine-tunes it, and attaches it to a situation. For example, the user is sitting in a movie theatre, and one of the services produces a loud notification. The unobtrusive adaptation space, defining the service’s adaptation behavior, makes sure the service is handled at the invisible obtrusiveness level in an *@quiet-place* situation (e.g., classroom).

However, mobile users are typically nomadic and move in a wide range of (previously unknown) environments, making it difficult even for them to foresee every situation-relevant environment (e.g., movie theatre). After quickly (and manually) turning off the device’s sound, the user selects the capture option. This option re-uses the screens from the previous “define” option (see previous section) and populates them, based on the user’s current environment. The user selects the location aspect (see Fig. 7.17), and sees that the “inside MovieTheatre” location

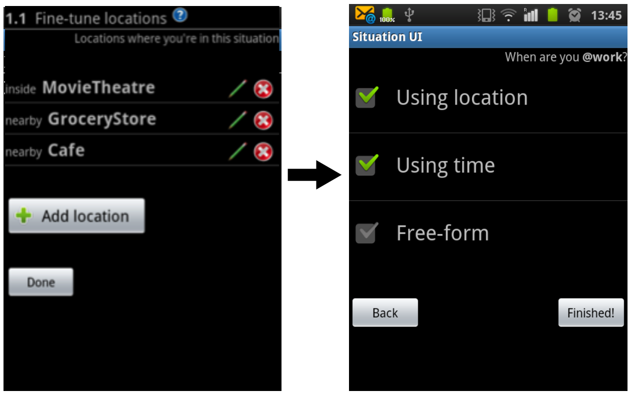


Figure 7.17: Interfaces to capture locations.

is present, as well as some other captured locations (e.g., "nearby Cafe"). S/he then removes the irrelevant locations, and also unchecks the time and free-form option, since they are not relevant in this case. In the final screen, the user attaches the fine-tuned context to the *@quiet-place* situation, thus making sure the invisible obtrusiveness level will be utilized in movie theaters as well.

7.4 Reinforcement evaluation

Learning systems have to be efficient and consistent with the real world in order to assure user satisfaction and acceptability of users. Thus, in order to measure the efficiency of our learning system and its correct functioning, we implemented a prototype and conducted an experiment. This prototype is composed of three components:

- *The learning component*: This component was deployed on a server side where the reinforcement learning algorithm was implemented. It was implemented in Java/OSGi.
- *The managed system*: This component was deployed in the mobile

device. It consisted on a notification system capable of receiving notifications from different services and delivering them to users. This system received notifications with the appropriate interaction resources to be used according to the obtrusiveness level. It was implemented on iOS 5.1 running on an iPhone 4. The interaction components available for delivering the notifications in this system were: vibration, loud and soft audio, speech, a badge icon, a dialog alert, and a banner. By means of these interaction components and a combination of them we were capable of giving support to all the obtrusiveness levels. We implemented a mobile application following a master-detail interface that displayed a master list of all the notifications received and the details for the currently selected notification. In this way, when a notification is received in the managed system, it is notified to the user using the appropriate interaction components for the current obtrusiveness level and is is also inserted in the master list of the application. By selection a notification of the list view, its details are shown. This notification system is further explained in Chapter 8.

- *A notification management system*: This component was developed to simulate the notifications from different ubiquitous services. This system supports the introduction of new notifications for the different services of the case study and the sending of them to the managed system. This component is also further explained in Chapter 8.

7.4.1 Case study description

For this test, we used a case study for supporting different ubiquitous services of the users' daily life, where different services were classified in a different obtrusiveness level according to an initial design and adapted then to a specific goal. This case study describes different ubiquitous services in a Smart Home such as a *shopping* service, a *healthcare* service, a *washing machine* service, an *agenda* service and a *recorder* service. Table 7.1 shows the services used and their initial obtrusiveness level designed.

Service	Initial obtrusiveness level
Washing machine	(slightly, reactive)
Healthcare	(aware, proactive)
Shopping	(slightly, proactive)
Agenda	(aware, proactive)
Video Recorder	(invisible, proactive)

Table 7.1: Initial obtrusiveness level of services

In particular, we defined a scenario for adapting several notifications of these services for a initially unemployed user profile. The notifications of the different services were adapted dynamically at different obtrusiveness levels according to users reaction. This allowed us to evaluate the efficiency of the learning system based on user feedback and the user experience with the system.

Initially, the services were defined in the initial obtrusiveness state mentioned. Then, due to the user starts working, some services were adapted because the received negative feedback. The service we are going to show in the evaluation is the *healthcare* service. It was initially designed in the *(aware, proactive)* obtrusiveness state because the service was important for the user. Then, the user changes his/her lifestyle because s/he starts a new job and s/he is working most of the day with other people and doesn't want other people to be aware of the notification. So, the notification is adapted to the *(slightly, proactive)* obtrusiveness state when the user starts to give negative feedback.

7.4.2 Evaluation procedure and results

The role of the user was taken by a human experimenter and consisted in receiving notifications in a given context and giving rewards to the system. For that, the experimenter had a goal behavior in mind and gave rewards mostly consistent with this goal. For example, the goal for the *healthcare* service was:

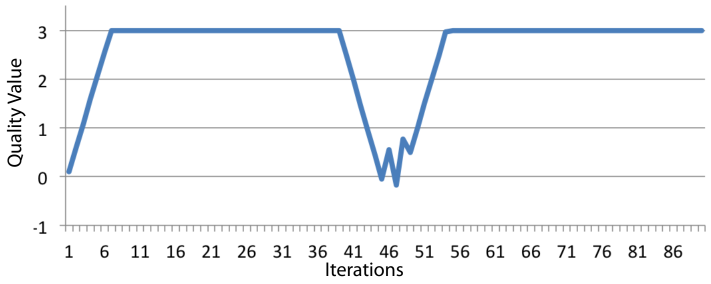


Figure 7.18: Quality values for the behavior of the *healthcare service*.

- To be satisfied with the initial design during 40 notifications.
- To change the mind and want to reduce the attention level of the notification due to start working.

The experimenter was also who grade the learned behavior (new obtrusiveness level) by indicating if the way to receive the notification was the one he had in mind or not. Fig. 7.18 shows the resulting quality values of the *healthcare* service behavior based on the rewards given by the experimenter for α=0'5, γ=0,99 and Δ=3.

The quality of a behavior is its correspondence to what user expects. The curve is rising or flat when the obtrusiveness behavior is optimal for the user (positive rewards). The quality drops when the experimenter changes his mind on the expected behavior (negative rewards) and it is fluctuating while the system is learning. The figure shows an initial phase where the experimenter was satisfied with the behavior (iterations 1 to 39). Then, the experimenter changed his mind and the curve drops (iterations 40 to 45). The system quickly learned the new behavior (iterations 46 to 49) and finally the user was satisfied after the new behavior was achieved (iterations 50 to 90).

It is worth noticing that the value of Δ is dependent on α and r because it defines more or less the number of iterations to learn a new behavior. If Δ is higher, the system will be less sensitive to anomalous rewards but a change in the user's mind will be slower to learn. On the contrary, if Δ is lower, the system will be more sensitive to anomalous rewards but will learn quicker. Thus, the choice of Δ value could be different for each user following this criteria.

7.5 Conclusions

In this work, we have presented a learning method for adapting interaction obtrusiveness based on user's feedback. Because user's needs and preferences can change over time and the initial interaction obtrusiveness design can not further satisfy the user, we introduce a system to adapt this initial design through experience. We exploit user feedback after receiving a notification in a certain obtrusiveness level in order to minimize the burden of mobile notifications and maximize user's satisfaction in a long-term use. In addition, the provided customization interfaces allow the users to re-define their own obtrusiveness preferences to support explicit adaptation over time.

8 Validation of the proposal

Measuring it!

"To measure is to know". "If you cannot measure it, you cannot improve it"
—Lord Kelvin, a.k.a. Sir William Thomson, n.d.

This chapter describes the evaluations performed to validate our approach by means of applying it in practice. By applying our approach, we want to illustrate how interaction can be adapted to provide an appropriate obtrusiveness level in an Ubiquitous Computing environment. Specifically, the present work has been validated from different perspectives according to the confronted research questions defined in Chapter 1:

Design method. The information captured at design should describe the aspects that are relevant to capture the interaction obtrusiveness adaptation requirements. This design method has to help designers in the specification of this kind of systems. Our research results show that the models defined are useful for designers to develop this kind of systems.

Requirements must be captured in a way that it is feasible to validate them with fast iterations. In this way, continuous feedback from users can be obtained to improve designs, since these designs have to be well accepted by the users and fit all their needs. Our research results show that fast-prototyping techniques can be applied in a way that reproduce the user experience of the final system and feedback from users can be easily mapped onto the models.

Self-regulating autonomic infrastructure. The provided infrastructure has to be subject to the same efficiency requirements as the rest of the system because the execution of the adaptation impacts overall system performance. Our research results show that our infrastructure gathers the necessary knowledge from the runtime models without drastically affecting the system response

In addition to objective measures, the attitudes of users using the system play an important role with regard to user experience. For this purpose, these subjective measures should be evaluated in a way that immerse users in a real environment. We evaluated the User eXperience after users interacted with our system, obtaining an enhanced user experience.

Learning system. The learning system has to be quick enough and consistent with the changing preferences of the users in order to guarantee user satisfaction with the system. Our research results show that it learns the new obtrusiveness preferences from user feedback with few iterations.

Also, the mechanisms and tools provided for changing and adapting the initial design and customizing the obtrusiveness according to each user preferences must allow end-users to define and update them with enough usability and expressivity. Our research results show the interfaces to be usable and expressive, allowing users to specify non-trivial situations and adaptations within a short time span.

In order to validate the above concerns, we put the approach in practice carrying out a case study based evaluation. It has been proven that case studies provide deeper understanding of the phenomena under study if proper research methodology practices are applied (Flyvbjerg, 2006). Specifically, we have developed a *Smart Home* case study and an *Adaptive Notifications* case study following the guidelines for case study research provided in (Runeson and Höst, 2009).

First, with the *Smart Home* case study we wanted to verify that the design method was appropriate and useful for describing the interaction obtrusiveness adaptation requirements in a way that help designers in their specification. In order to respond to this, we modeled the case study and compared our design method to the traditional method (hand-coding). Then, we made use of fast-prototyping techniques in order to evaluate the user satisfaction with the designed adaptations and re-design the models. On the one hand, feedback was gathered from end-users in order to verify how the design method could deal with the changes iteratively and the degree of user satisfaction with the designed system. On the other hand, the cognitive workload was measured in order to analyze how our system is capable of handling the attentional resources of users by means of the adaptations.

Then, with the *Adaptive Notifications* case study we wanted to validate the User eXperience by using our system. In order to evaluate this, we developed an adaptive notifications' management system capable of receiving notifications from different ubiquitous services and delivering them to the users using the most appropriate interaction resources according to the user situation. This allowed us to identify the relevance of the obtrusiveness adaptation in the user experience for the interactions. The user experience was measured by comparing our adaptive system with a non-adaptive one. Also, with this case study we validated the usability of the customization interfaces for end-users by means of a user evaluation. In this experiment, users had to interact with the interfaces for defining and changing different situations and obtrusiveness adaptations.

In addition, the scalability of the self-regulating system was evaluated in Chapter 6, and the efficiency of the learning system was validated in Chapter 7, obtaining satisfactory results in both evaluations.

We also have applied our approach in other domains by means of different case studies such as a *Smart Hotel*, a *Smart Library* and a *Home-Care*. This allowed us to identify the advantages and limitations when applying our approach. We evaluated both, the experience of the designers with the design method proposed and the experience of the end-users with the defined interaction adaptations.

The remainder of this chapter is structured as follows. Section 8.1 introduces the *Smart Home* case study and the evaluations performed using this case study. Section 8.2 describes the *Adaptive Notifications* case study and provides information regarding the experiments performed using this case study. After these two sections, in Section 8.3, we give details about our experiences applying our approach in other domains. Then, we analyze the advantages and problems found during the application of the approach in Section 8.4. Finally, Section 8.5 concludes the chapter.

8.1 Smart Home case study

We applied our approach to a case study of a smart home environment based on the scenario of service adaptation we developed in (Cetina et al., 2009). We extended the services defined in the original case study in order to adapt the obtrusiveness level at which they are presented to the user.

8.1.1 Design of the case study

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

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

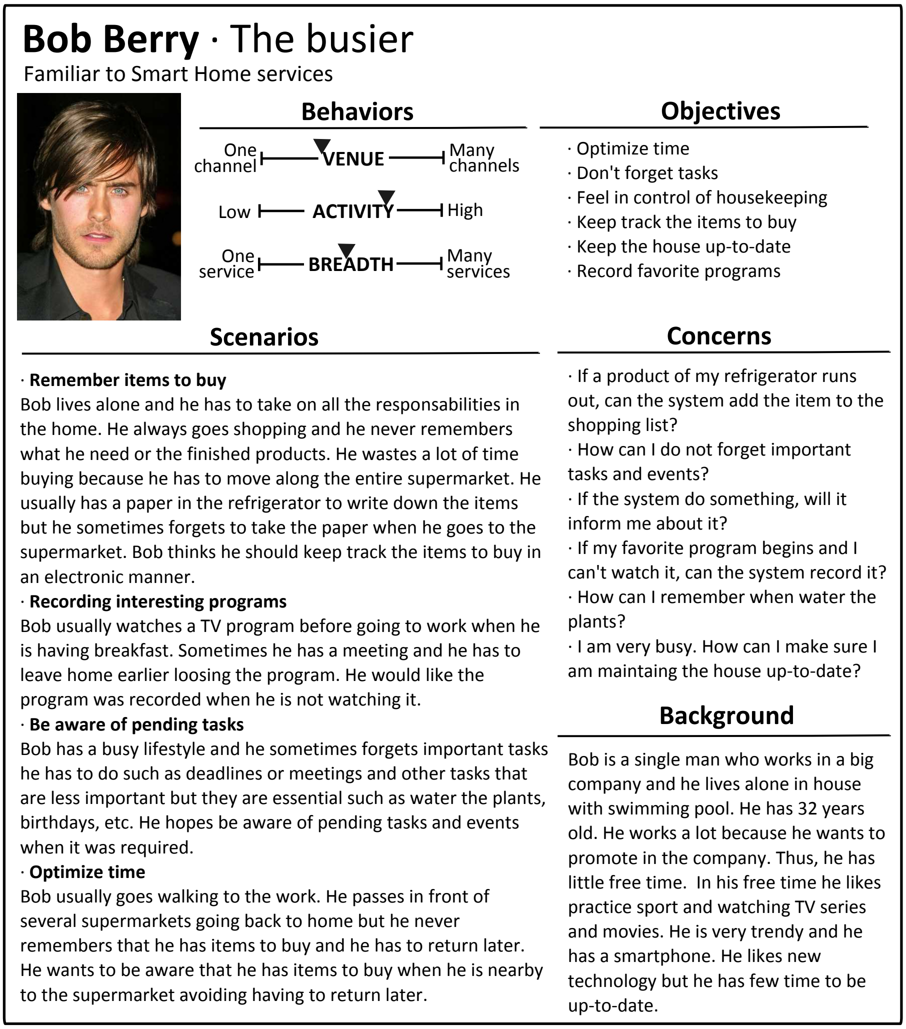
When he was going back to home, he was nearby to a supermarket and the mobile notified him about it, showing the map to arrive to the supermarket. When he was there, the map was changed by the floor map of the supermarket. At the same time, the mobile suggested him the items to the shopping list that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

For the design of the services defined in the case study, we applied the design method defined in Chapter 5. According to this method, different aspects of the services should be captured in models.

8.1.2 Applying our design method

According to our design method, the different aspects of the interaction obtrusiveness adaptation should be captured in models. The detail of the information captured in each modeling perspective includes the *persona definition* in order to understand the users and detect *the services and obtrusiveness*, the *obtrusiveness modeling* for each service detected, the *interaction variability modeling*, and the *concrete interaction modeling* that determines the final user interface provided. All these aspects are detailed below.

Persona definition



Bob Berry · The busier

Familiar to Smart Home services

Behaviors

One channel ⊢ VENUE ⊣ Many channels

Low ⊢ ACTIVITY ⊣ High

One service ⊢ BREADTH ⊣ Many services

Objectives

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

Scenarios

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

Concerns

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

Background

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

Figure 8.1: A detailed persona

In order to give a clear picture of how users are likely to use the system and what they will expect from it, we define personas. Personas capture relevant information about customers that directly impact the design process: user goals, scenarios, tasks, functionalities, and the like. Figure 8.1 shows the description of the persona for Bob.

Services and obtrusiveness definition

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

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

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

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

Plant watering. Users have a busy lifestyle and they usually forget to water their plants. This service is in charge of remind and tell users that their plants need water and water the plants automatically when they need it.

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

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

The contextual information relevant for the given persona is:

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

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

User engagement in other activities. This is an important factor to be taken into account for service adaptation. For example, if the user is engaged in an important activity (e.g., working, in a meeting), a service will be presented in a subtle manner avoiding to disturb him. However, if the user is in his free-time, a service will demand his complete attention.

Once we have defined the personas, the important services for the persona and the relevant contextual information (to form the user situations) in which the adaptation can depend on, we define the way in which services are presented in terms of obtrusiveness for the case study. This information is detailed below.

Obtrusiveness modeling

Each service can be provided at different obtrusiveness level depending on the user situation in which the service is performed. Figure 8.2 shows the services of the *Smart Home* case study and their obtrusiveness level

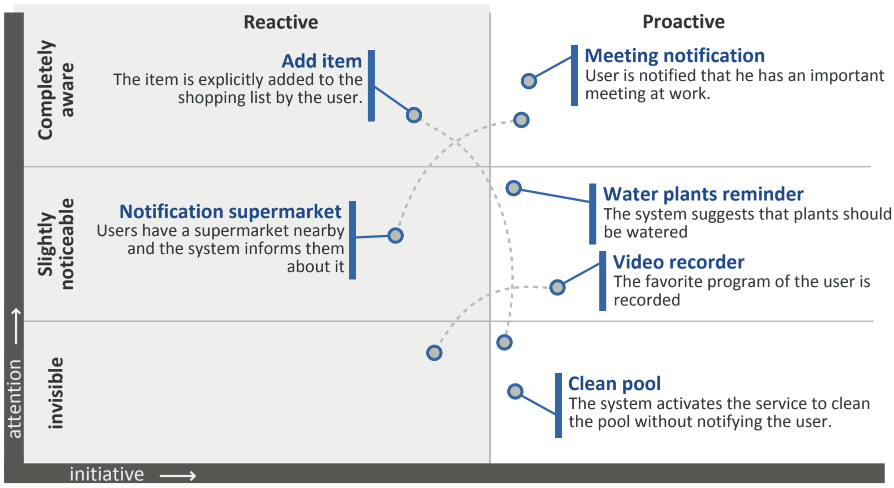

Reactive
Proactive
Completely aware
Slightly noticeable
invisible
attention
initiative
Add item
The item is explicitly added to the shopping list by the user.
Meeting notification
User is notified that he has an important meeting at work.
Notification supermarket
Users have a supermarket nearby and the system informs them about it
Water plants reminder
The system suggests that plants should be watered
Video recorder
The favorite program of the user is recorded
Clean pool
The system activates the service to clean the pool without notifying the user.

Figure 8.2: Obtrusiveness level defined for each service in the *Smart Home* case study.

for Bob. The unobtrusive adaptation space in this case was defined by dividing each axis in different parts as it was illustrated in Chapter 5. The attention axis is divided in three levels depending whether the interaction should be *invisible* to the user, *slightly noticeable*, or *completely aware* for the user. The initiative axis is divided in two parts that represent interactions initiated by the user (*reactive*) and interactions initiated by the system (*proactive*).

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

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

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

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

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

Supermarket notification. When the user is in the proximity of a supermarket (user location), he is informed about a supermarket nearby. Depending on the distance to the supermarket and the number of items on the shopping list, the notification will be different. On the one hand, if the user is closer to the supermarket, the system *proactively* will notify the user in an explicit manner (the user is *completely aware*). On the other hand, if the user is far, the notification will be *slightly noticeable*. Then, when the user is in the supermarket, the system shows the items to buy that are in the shopping list at the *slightly noticeable* level of attention.

Clean pool. The system *proactively* activates the service to clean the pool in an *invisible* manner for the user due to the user does not have the mobile nearby (user situation without the mobile).

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

Interaction variability modeling

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

In (Chittaro, 2010), Chittaro summarizes the many channels that can be exploited to send and receive information from a mobile device. For this case study, we have defined the feature model utilizing these modalities. To describe the constraints between modalities we have followed the study presented in (Lemmelä et al., 2008) to identify modalities and modality combinations best suited for different situations and information presentation needs.

Figure 8.3 shows the decomposition of the available interaction in the feature model and the constrains for their selection. We have divided the interaction into groups of expression, visual, auditory and haptic modalities. These four main features include a set of manifestations of input and output modalities.

Then, we have to choose for each service the interaction features that are going to support the obtrusiveness level defined. Table 8.1 shows a view of the interaction analysis results performed for the services. The table shows for each service, the obtrusiveness level at which

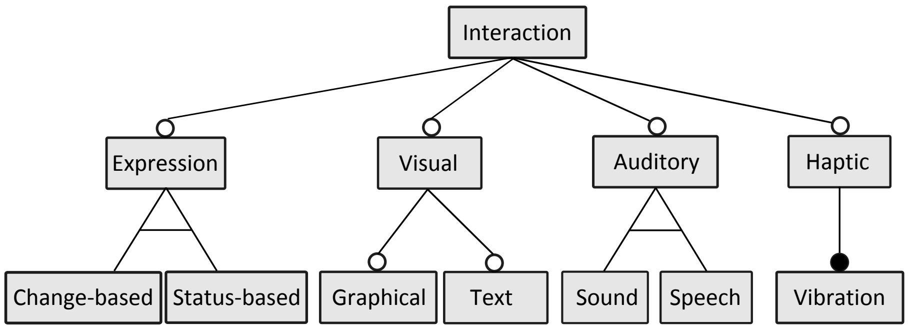


Figure 8.3: Decomposition of interaction aspects using the Feature Model.

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

Table 8.1: Interaction features for each service in the unobtrusive adaptation space

it can be performed and the interaction features selected for the obtrusiveness level. The services that have more than one obtrusiveness level associated is because they depend on the user situation and can change at runtime.

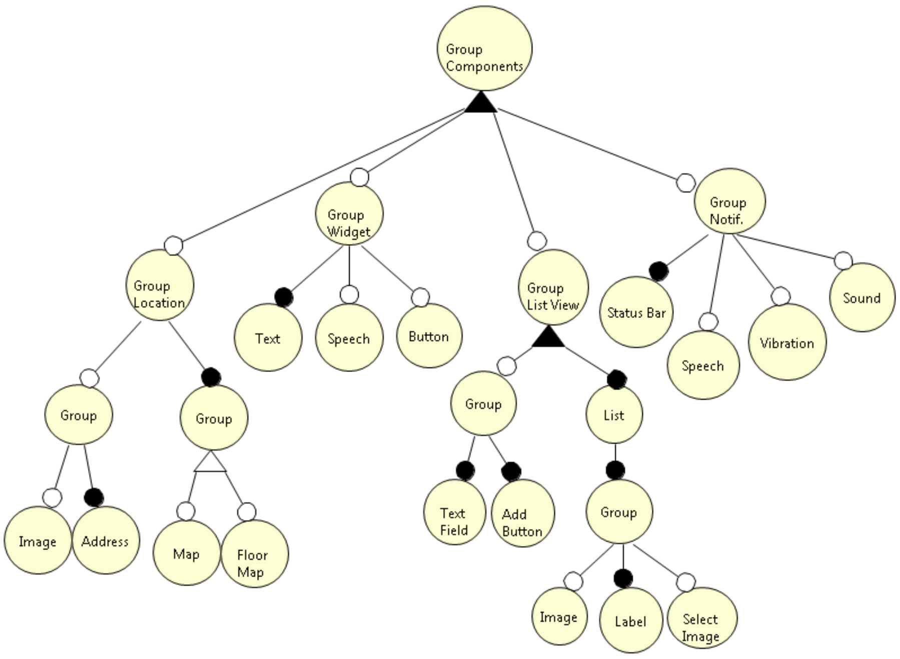


Figure 8.4: Concrete UI components of a Smart Home system.

Concrete interaction modeling

In order to define the concrete user interface components that support the interaction techniques available, we define the node tree of our system. For the *Smart Home* case study, the concrete components are shown in Figure 8.4.

The concrete UI components that support the different interaction features are specified in Table 8.2

The requirements captured following our design method describe the interaction obtrusiveness adaptation of services to support the *Smart Home* scenario. Since the design method was followed, the different decisions are organized in different layers and consistency is guaranteed

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

Table 8.2: Linking between interaction features and concrete components

among them. However, in order to validate our approach, we have performed an experiment to measure the benefits that introduce our model-based solution for designers (i.e., its usability) over traditional development. This evaluation is detailed below.

8.1.3 Evaluating the design method

In order to measure the **usability of our approach**, we have followed an empirical framework to perform usability evaluations for MDD tools (Panach et al., 2011). According to this framework, the usability of a MDD tool is evaluated by means of: satisfaction, efficiency and effectiveness. In this work, we focus our study on the satisfaction and the efficiency. The aim of this evaluation is to compare the usability measure obtained by our proposed approach (model-based development) over the traditional development (hand-coding development).

According to the Goal/Question/Metric template (van Solingen and Berghout, 1999) the objective of the experiment was:

Analyse	our model-based solution
For the purpose of	evaluating its usability
With respect to	the traditional design and development (hand-coding)
From the viewpoint	of the designer
In the context of	computer science developers

From this objective, the following null hypotheses were derived (one for satisfaction and another for efficiency) and the two types of variables were identified:

Null hypothesis 1, $H1_0$: The satisfaction when following our method for developing non-disturbing adaptive interactions is the same as the traditional design.

Null hypothesis 2, $H2_0$: The efficiency when following our method for developing non-disturbing adaptive interactions is the same as the traditional design.

Identification of variables

We identified two types of variables:

- **Dependent variables:** In this work, usability was the target of the study, which was measured in terms of **satisfaction** (measured with respect to the designers' perceptions of usefulness and overall satisfaction) and **efficiency** (measured by the task completion percentage in relation to the time spent to perform a task).
- **Independent variables:** The development method was identified as a factor that affects the dependent variables. This variable had two alternatives: (1) model-driven development (MDD) and (2) traditional development.

Experimental context

Experimental subjects. Eight subjects participated in the experiment, all of them being researchers in software engineering. Their ages ranged between 25 and 40 years old. The subjects had an extensive background in Java programming and modeling tools.

Objects of study. The experiment was conducted using the *Smart Home* case study detailed above. In order to shorten the evaluation process for both development methods and to achieve similar implementations from user to user, we provided the subjects with an example service to guide the development of the other services. Specifically, we provided them with a traditional implementation of the *shopping list* service as well as its modeling using our method.

Instrumentation. The instruments that were used to carry out the experiment were: (1) a *demographic questionnaire* to know the level of the users' experience in Java programming and modeling tools, (2) a *task description* with the tasks that the users must carry out, and (3) a *survey* with a list of questions defined to capture the duration times of each task and the users' perceptions in a 7-Likert scale format. Specifically, the tasks that the subjects had to perform were the development of the *Smart Home* case study using our model-based solution and the traditional one. More detail about the instruments can be found in Appendix B.

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. This threat was minimized with a demographic questionnaire that allowed us to evaluate the knowledge and experience of each participant beforehand. This questionnaire revealed that most users had experience in Java programming and modeling techniques. This threat was also minimized by providing the subjects with the *shopping list* service, which helped and guided them in the development of the rest of the services. Also, our experiment was threatened by the *reliability of measures* threat: objective measures, that can be repeated with the same outcome, are more reliable than subjective measures. In this experiment, the precision of the measures may have been affected since the activity completion time was measured manually by users using the computer clock. In order to reduce this threat, we observed subjects while they were performing the different activities in order to guarantee their exclusive dedication in the activities and supervise the times that they wrote down.

- **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 dividing the experiment into different activities.

- **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 behaviour on their guesses. We minimized this threat by hiding the goal of the experiment.

- **External validity:** This type of validity concern is related to conditions that limit our ability to generalize the results of the experiment to industrial practice. Our experiment might suffer from *interaction of selection and treatment*: the subject population might not be representative of the population we want to generalize. We used a confidence interval where conclusions were 95% representative. This means that if conclusions followed a normal distribution, results would be true for 95% of the times the evaluation would be repeated.

Experimental design procedure

We followed a *within-subjects design* where all subjects were exposed to every treatment/method (MDD and traditional development). The main advantage of this design was that it allowed statistical inference to be made with fewer subjects, making the evaluation much more streamlined and less resource heavy (Wohlin et al., 2000). In order to minimize the effect of the order in which the subjects applied the methods, the order was assigned randomly to each subject. Also, we had the same number of subjects starting with the first method as with the second in order to have a balanced design. In this way, we minimized the threat of learning from previous experience.

The study was initiated with a presentation in which general information and instructions were given. Next, the experiment started with a demographic questionnaire to capture the user's background. Afterwards, the task description and the survey were given to the subjects and they started to develop the case study following the two kinds of development (MDD and traditional) in the indicated order to each user. For each task of the development, they filled in the survey to capture the development times and their perceived satisfaction. The task completion percentage was obtained by deploying the developed adaptive behavior and testing it. As in the survey subjects wrote down the time at which they started and completed the task, efficiency was derived using these times. Furthermore, the perceived satisfaction was captured using the two perceptions of satisfaction (system usefulness and overall satisfaction) of the CSUQ questionnaire (Lewis, 1995).

Analysis and interpretation of results

A *paired t-test* was used to compare the mean of the subjects' means to evaluate if there was a difference in overall usability between the two methods: MDD versus traditional. Statistical analysis has been carried out using the IBM SPSS Statistics V20 at a confidence level of 95% (α=0.05) (Bruin, 2011). When the critical level (the significance) is higher than 0.05, we can accept the null hypothesis because means are not statistically significantly different. Table 8.3 presents the descriptive statistics for each of the studied measures.

Measurement	Method	Mean	N	Std. deviation	Signif.
Satisfaction	Traditional	4,81	8	2,78	,000
	MDD	2,33	8	1,71	,000
Efficiency	Traditional	0,47	8	0,55	,000
	MDD	2	8	0,48	,000

Table 8.3: Descriptive statistics for each measurement

Regarding the *perceived satisfaction*, designers were more satisfied with the model-based approach. A low value in the mean column of Table 8.3 implies a good perception of overall satisfaction, while a high value implies a bad perception. As the table indicates, there is a difference between the pair of means of the satisfaction measure. Since the MDD method provided a lower mean, subjects were more satisfied with it. Also, more dispersion was found in the perceived satisfaction with the traditional development (std. deviation=2,78). This is because subjects that had more experience in Java programming had less problems in the development and their perceived satisfaction was higher than the others. The significance of the paired t-test for the satisfaction means was 0.000, which means that we can reject the null hypothesis H_{10}, therefore, the perceived satisfaction using both methods is different. As the results are better in the MDD method, we have a strong evidence that **the perceived satisfaction when using our method is better than when using the traditional one**.

With regard to *efficiency*, we obtained a significant difference in the mean values. According to the analysis that was carried out for efficiency, H_{20} can be rejected. As Table 8.3 shows, the mean for efficiency was 0,47 for traditional development and 2 for MDD. Therefore, efficiency was significantly better with the MDD. In general, the subjects completed all the tasks for both methods; however, the development time using the traditional development was higher than following the MDD method. Regarding the standard deviation, it was low for both development methods indicating that development times tended to be close for each development method. These results provide strong evidence that **the efficiency when following our method is better that when following the traditional one**.

8.1.4 Simulating the design

The MDE techniques defined in Chapter 6 can be applied to the design models to obtain a software solution to support the system. However, there is still place for fast-prototyping. In the case of adaptive ubiquitous services, deployment efforts are high if real hardware is used. Designers need certain guaranties that the adaptation defined will fit well when it is finally deployed.

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

Questionnaire and participants

Once the user is immersed in the simulated environment, the user satisfaction in terms of usability and interaction adaptation is evaluated. To evaluate these aspects, we used an adapted IBM Post-Study questionnaire (Lewis, 1995) in conjunction with the questionnaire defined by Vastenburg et al. in (Vastenburg et al., 2008). On the one hand, IBM Post-Study is a questionnaire that measures user satisfaction with system usability. On the other hand, some questions were taken from the Vastenburg questionnaire to evaluate home notification systems such as messages acceptability and interaction adaptation. The three dimensions evaluated in our questionnaire were: *usability of the system*, *messages acceptability* and *interaction adaptation*.

The first dimension focuses on measuring users' acceptance with the usability of the system; the second one focuses on the general acceptability considering the messages and the user activity at the time of notification; and finally, the third dimension is about users' satisfaction in the interaction adaptation.

Also, we included a NASA task load index (TLX)[1] test. This test assesses the user's subjective experience of the overall workload and the factors that contribute to it on six different subscales: *Mental Demand, Physical Demand, Temporal Demand, Performance, Effort*, and *Frustration*.

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

Procedure

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

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

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

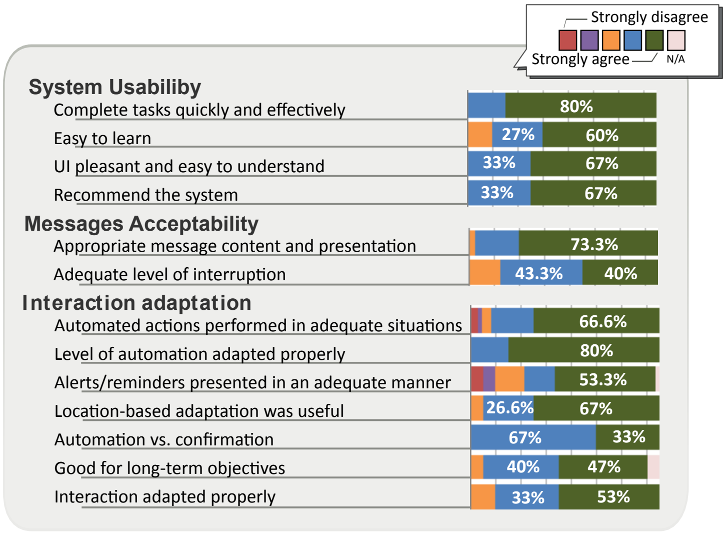


Figure 8.5: Summarized results.

final system with minimal effort.

Results of user satisfaction evaluation

Figure 8.5 shows a summarized table of the obtained results[2].

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

With regard to the *messages acceptability*, the results were also positive, but more dispersion was found in them. This was due the different perception each user had about what was considered to be a relevant or urgent message. Although participants had to adopt the personas

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

role and adjust their behavior to the personas needs, this is difficult when they have another needs. In the study made by Vastenburg et al. (Vastenburg et al., 2008), they pointed out that the more urgent the message was considered to be, the higher the level of intrusiveness should be. In our results, the content and presentation of the different messages were considered appropriate by the 73% of the subjects. Some users (20%) found some services to be intrusive, but the interruption level was in general (80%) considered adequate to each situation.

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

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

Results of the workload evaluation

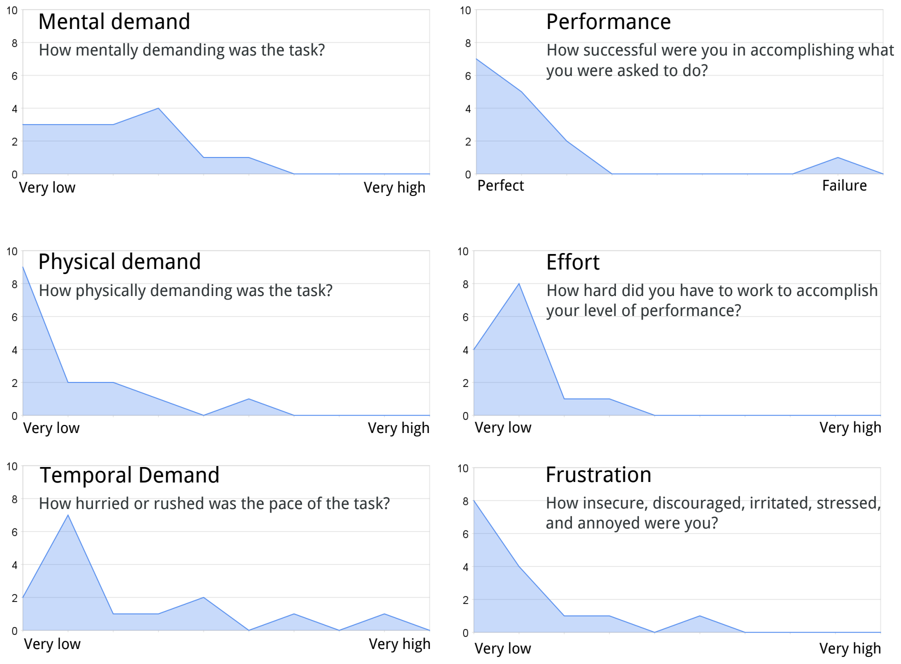


Figure 8.6: Nasa TLX results

The results on workload are shown in Figure 8.6. We show each subscale in a different diagram. The *Mental Demand* diagram shows that not all the tasks were simple and easy. Mostly, mental demand was low but some tasks in the experiment required more attention, increasing the mental demand. Some users would prefer more automation in the services. *Physical demand* was low excepting for the services that required more attention. Moreover, some users were not familiar with the use of a smartphone. For these users, the physical demand was higher.

The low workload was accompanied by good performance. The majority of users could accomplish the goals of the tasks proposed (see the *Performance diagram*) without much effort (see the *Effort diagram*) and with a low degree of frustration (see the *Frustration diagram*). *Tem-*

poral demand did not provide any significant results since the results are very scattered. This results show that users did not understand the question very well.

Adaptation re-design

The feedback from users was used to re-design the adaptation of the services iteratively. Some concerns resulted in minor modifications of the mock-up interfaces (e.g., using graphical metaphors and bigger buttons). Other suggestions were more relevant to the process redesign since they involved changing the *level of obtrusiveness* for some tasks and providing more contextual information.

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

The context conditions associated to the *water the plants reminder* were also problematic. In the first iterations of the design, this reminder was in a *slightly-noticeable* level of attention with the condition of using vibration if users were engaged in other important activities. In the case study, this reminder appeared during the meeting. However, users wanted this reminder only appeared when the user was at home.

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

8.2 Adaptive Notifications case study

We developed a case study for supporting different ubiquitous services of the users' daily life. The case study was focused on the notification mechanisms used in different situations since the adaptation of attentional resources is a key factor in the user experience of this kind of services (Vastenburg et al., 2008). In particular, we defined a scenario for adapting several notifications of one day in the context of a university professor. The notifications of the different ubiquitous services were adapted dynamically at different obtrusiveness levels depending on the user context. This allowed us to identify the relevance of the obtrusiveness level in the cognitive load and user experience for the interactions.

8.2.1 Design of the case study

The scenario focuses on a daily day of a university professor named Matt. He lives in a smart home with ubiquitous services with his wife and his son. Every day, he gets up at 7 a.m. and took a shower. While he was in the bathroom, the washing machine notified him that it was full load and ready to start. Because he had the mobile in his room, the notification was presented in a slightly level of attention since he was not going to be aware of it at that moment. Then, during the breakfast, the healthcare service reminded him to take the vitamins. Due to he was alone with the mobile on the table, the notification appeared in the completely-aware level of attention by means of speech. When he was leaving home to go to work, the weather service suggested him to take the umbrella. As he had the mobile at hand and he was alone, the notification was presented at the highest level of attention. However as the weather service had a medium priority for him, a dialog was used to present the notification instead of speech.

While he was driving to work with a workmate, the agenda service reminded him a meeting to attend at work. Because he was driving with company, the notification was presented in a slightly level of attention. At work, Matt was having coffee with his workmates and the facebook service notified a post on his timeline. The notification appeared in a slightly level of attention by means of vibration and a banner for not disturbing the conversation. During the meeting, the agenda service reminded Matt about a deadline approaching. Because he was in the meeting, the notification appeared in an invisible manner for not interrupting the meeting. After the meeting, he was having lunch with his workmates, and the healthcare service reminded him that he had to take the pills. This notification was presented in a slightly manner due to the privacy of the message. In the afternoon, Matt was giving a course and two notifications were suggested to him requiring different levels of attention. One from the facebook service in an invisible manner due to the facebook service had low priority, and another from the home messages in a slightly manner because of their urgency.

When he was going back to home, he was nearby of a supermarket and the mobile notified him about an item to buy. Because he was alone and there was an urgent item in the shopping list, the notification was totally aware using the speech of the car. When Matt arrived at home, he was watching the TV and the washing machine remembered him to get out the laundry at the highest level of attention because he was not engaged in an important task. Then, Matt went running listening to music with the headphones. The notification of the agenda about the deadline approaching increased the attention level because Matt was not aware of it in the morning. This time, the notification was presented in the aware level of attention. At the end of the day, Matt was sleeping and the shopping service suggested him an item to put in the shopping list. Due to he was sleeping, the notification was invisible for not disturbing him.

Appendix B describes in detail the design of the *Adaptive Notifications* case study in the same way we have described the *Smart Home* case study. This description comprises the designed models that conform the ubiquitous services of the case study by means of applying our design method.

8.2.2 Evaluating the User Experience

This experiment was focused on evaluating the User eXperience (UX) of users using our system (to evaluate their feelings). This evaluation has become important in mobile contexts given the ubiquity and intelligent capabilities of these kind of systems. For the evaluation, we presented to users a non-adaptive version of our system (usual notification system) and the adaptive one (our system) based on the proposed scenario. In this way, users could compare both systems and better measure their UX. At the end, users were handed a questionnaire to collect their attitudes towards our system.

Experimental design

The goal of the scenario was to show users the capabilities of our self-regulating system by emphasizing the following points:

- The notifications' interaction is adjusted according to the user attentional resources in each situation (e.g., when the user is in a meeting the notification is presented silently).
- In the same situation, different services can be presented in different obtrusiveness levels according to user preferences (e.g., when the user is giving a course, the facebook and home messages are presented in a different obtrusiveness level).
- Different services in the same obtrusiveness level can be presented in a different way according to their priorities for the user (e.g., the agenda and the weather service in the aware obtrusiveness level are presented in a different way because they have different priorities).

Table 8.4 shows the notifications of the services delivered to the user in the proposed scenario in order of appearance, the user situation in the time of the notification and the obtrusiveness of the notification for that context. For the experiment, services were classified in a different unobtrusive adaptation space according to three priorities for the user profile: high, medium and low priority. Specifically, the *healthcare*, *agenda* and *home messages* services were given a high priority; the *washing machine* and the *shopping* the medium priority; and finally, the

Service	**User situation**	**Obtrusiveness**
Washing Machine	Mobile in other room	(slightly, reactive)
Healthcare	Having lunch, alone mobile on the table	(aware, proactive)
Weather	Leaving house, alone, mobile at hand	(aware, proactive)
Agenda	Driving with company	(slightly, proactive)
Facebook	Having coffee with company	(slightly, proactive)
Agenda	In a meeting	(invisible, proactive)
Healthcare	Having lunch, with company	(slightly, proactive)
Facebook	Giving a course	(invisible, proactive)
Home Messages	Giving a course	(slightly, proactive)
Shopping	Driving alone, near a supermarket	(aware, reactive)
Washing Machine	Watching TV, alone in home	(aware, proactive)
Agenda	Running with headphones	(aware, proactive)
Shopping List	Sleeping	(invisible, proactive)

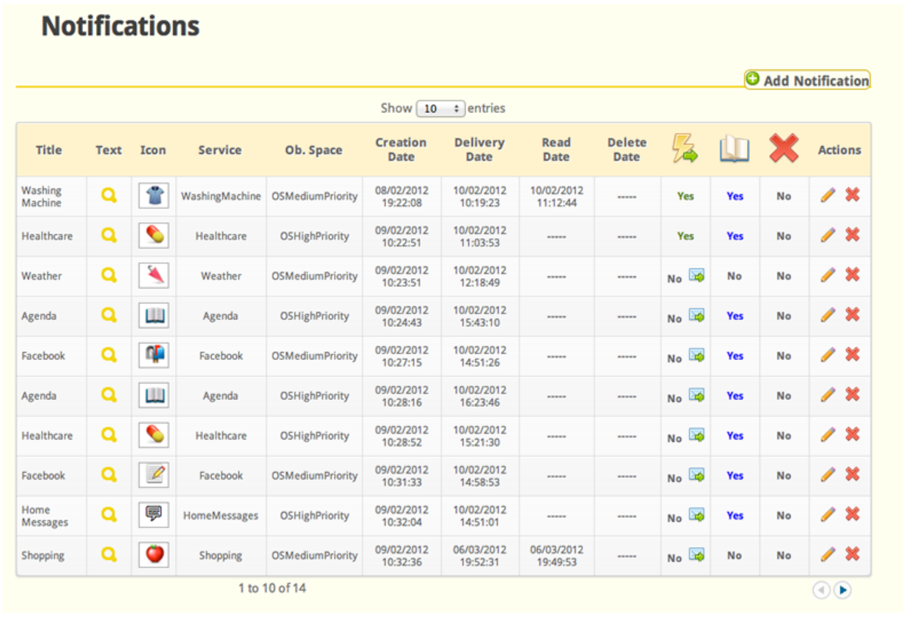
Table 8.4: Notifications received during the experiment.

facebook and the *weather* services were in the low priority. It is worth noticing that the obtrusiveness of services was adjusted to a professor profile and his needs.

Testbed

The managed system used for the test was running on an iPhone 4 (iOS 5.1). The interaction components available for delivering the notifications in this system were: vibration, loud and soft audio, speech, a badge icon, a dialog alert, and a banner. By means of these interaction components and combinations of them we were capable of giving support to all obtrusiveness levels.

For delivering the notifications to subjects in the opportune context,



Notifications

Add Notification

Show 10 entries

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

Figure 8.7: Notification management system.

we implemented a notification management system. This system supports the introduction of notifications for the different services and the sending of them to the managed systems. A screenshot of this system is shown in Figure 8.7. Using this system, we were able to simulate the notifications from pervasive/mobile services in order to deliver them at the context described in the case study.

On the server side, the target platform 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.

Questionnaire and participants

Fifteen people participated in this experiment (9 men and 6 women). Their age ranges from 19 to 50. Most of them were university students (master, or PhD) and daily mobile users. Each subject performed the defined scenario throughout a day. First, participants were given a short description of the scenario situations and the notifications presented in each situation. Then, they watched a video[3] of the proposed scenario (following the *video creation* step of our simulation phase) to clarify the description and, finally, they used it in a real environment.

To measure the UX, we followed one of the most influential models proposed by Hassenzahl (Hassenzahl, 2008). According to this model, each interactive system has a pragmatic (related to usability) and hedonic (related to users' self) quality that contribute to the UX. Based on this model, we use the AttrakDiff 2 questionnaire (Hassenzahl, 2008) to measure UX. The questionnaire consists of twenty-one 7-point items with bipolar verbal anchors. 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) related to personal growth of the user and the need to improve personal skills; *Hedonic Quality Identification* (HQ-I) 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.

Results of the user experience evaluation

Fig. 8.8 illustrates the mean values of the UX dimensions for both systems[4]. According to the results, we observe considerable deviation between the instances in HQ-S and ATT. In HQ-S, mean scores of the non-adaptive version were -1.25, and 2 for the adaptive one. In ATT, mean scores of the non-adaptive version were -0.25, and 2.4 for the adaptive one. These values indicate that users perceived a huge difference of the hedonic quality (stimulation) and overall appeal (attraction) between both versions, considering our system better in these aspects.

[3]The video can be found on: http://www.pros.upv.es/adaptio/Notifications.mov
[4]Complete results in http://dl.dropbox.com/u/14910519/uxresults.pdf

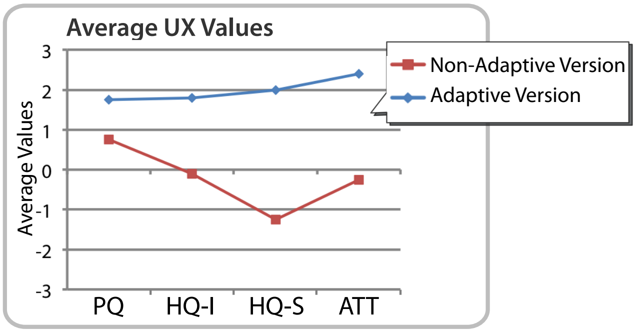


Figure 8.8: Mean UX values for both systems.

Regarding the PQ, marginal deviation has been observed indicating that this factor does not significantly influence in the UX of the different versions. Even so, PQ results were higher in our system, indicating that users considered our system more usable. 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.

Regarding the mean values of the word pairs, there are some of particular interest in our research. Results indicate 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 to interrupt the user. Users rated the adaptive system a bit more *complicated* since it has more configuration features to take into account compared to the non-adaptive, which only has basic configuration features. Also, users appreciated our system a little more *unpredictable* compared to the non-adaptive, due to they unknown initial designed adaptations.Despite these detected issues, the adaptive version was considered more *innovative*, *challenging* and *attractive*.

Thus, results points out that with our obtrusiveness adaptations, **user experience is enhanced**.

8.2.3 Evaluating the customization interfaces

We validated the usability and expressivity of the customization interfaces by means of a user evaluation, where users had to change the obtrusiveness adaptation behavior via the obtrusiveness personalization interface, and specify six situations of varying complexity via the definition and capturing options of the user specification interface.

Experimental design and procedure

In order to evaluate the *Obtrusiveness Personalization Interface* the users had to perform the following tasks:

- Change the interaction resources associated with a specific obtrusiveness level.
- Edit an existing transition between two obtrusiveness levels (change the user situation that triggers the transition).
- Create a new transition between two obtrusiveness levels.
- Delete an existing transition between obtrusiveness levels.
- Edit the information of an existing user situation.
- Delete an existing user situation.
- Create a new user situation.

Regarding the evaluation of the *User Situation Specification Interface*, the users had to specify six situations of varying complexity, using the definition and capturing options. Below, you can find the list of situations. Note that, since "capturing" the first five situations proved trivial, the users only had to use the capture option for the sixth, most complex situation.

- @work (1): You are inside the "DSIC" building (an educational building, with URI http://dsic.upv.es), during working hours (9AM – 18PM, from Monday to Friday).

- @meeting: You are inside a meeting room during work hours (9AM – 18PM).
- @quiet-place: You are inside a movie theatre or a theatre (time independent)
- @teaching: You are inside a classroom, during 13-17h on Monday, 11-13h on Tuesday, and 13-17h on Friday.
- @free-time: The current time on weekdays is from 18PM to 8AM, and the whole day during the weekend.
- @work (2): You are inside your office (which is linked to you via the housesPerson property), during working hours (9AM – 18PM). You can assume that http://you.upv.es is your personal URI. You will have to utilize the free-form method to express this situation.

In each user session, we took five minutes to shortly explain the interfaces, and then let the user perform the described tasks. We noted the required time, as well as any encountered difficulties and errors during their tasks. After performing their tasks, the users filled out the questionnaire.

Questionnaire and participants

In order to evaluate the efficiency of the customization interfaces, we used the Post-Study System Usability Questionnaire (PSSUQ) (Lewis, 1995). This questionnaire is a 19-item instrument for assessing user satisfaction with system usability. Specifically, it studies the following four dimensions: *overall satisfaction* with the system, its *usefulness*, *information quality*, and *interface quality*.

A total of 8 subjects participated in the experiment (5 male and 3 female), between the ages of 25 to 35. All of them had a strong background in computer science, being students or researchers; they were also familiar with the use of a smartphone, and 4 out 8 owns an Android-based smartphone similar to the one used in the experiment.

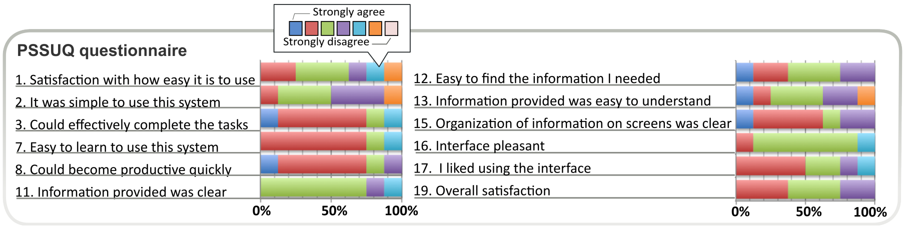


Figure 8.9: Summarized questionnaire of resutls.

Results of the customization interfaces evaluation

Fig. 8.9 shows a summary of the PSSUQ questionnaire results; the complete dataset can be downloaded from http://wise.vub.ac.be/Mobiquitous2012/. Overall, users found the interfaces simple to use (questions 1, 2) and very easy to learn (7), while they also felt they could complete tasks effectively (3) and quickly become productive using the system (8). Users found the provided information more or less clear (11), easy to find (12) and understand (13), and clearly organized (15). Overall, around 80% of the users found the interfaces pleasant (16), and 75% liked using the interfaces (17). Averaging the questionnaire results, on a scale from 1 (strongly agree) to 7 (strongly disagree); overall satisfaction was 3.09, usefulness was 3.2, information quality was 3, and interface quality was 3.04.

On average, users took about 10 minutes to personalize the obtrusiveness and 13 minutes to specify all user situations; since "capturing" the first five situations proved trivial, they were left out the remaining evaluations of the capture option. Regarding the *obtrusiveness personalization interface*, users found more difficulties in the tasks to manage the conditions. This was because they were less familiar with these aspects. However, they could complete all the tasks effectively. With regards to the *situation specification interface*, one user had initial difficulty with the location method; but the other users had no problems. Six out of seven users had difficulty using the free-form method while testing the define option; on the other hand, they found using this method easier when “capturing” situations.

In conclusion, the evaluations show the interfaces to be usable and expressive, allowing users to change the obtrusiveness design and specify non-trivial situations within a short time span. Moreover, the capture option makes it very simple to specify most situations. The free-form method, which allows for more generic and complex situation definitions, proved to be more difficult to use and have a steep learning curve. However, using this option becomes much easier in the capture option, since users are able to fine-tune a given free-form specification instead of creating one from scratch. We do note that this is a preliminary evaluation, and additional experiments, with a larger and more heterogeneous user group, are needed to confirm and generalize these results.

8.3 Experiences applying our approach

This section analyzes the experience of applying our approach in different case studies of other domains. These experiences have served to determine whether (1) the design method was adequate to represent and manage the knowledge involved in the design of interaction obtrusiveness adaptation, (2) the obtrusiveness is an issue that really affects users when interacting with a system, and (3) the adaptation performed in the different domains help to manage the user’s attentional resources.

A brief description of the case studies and the obtained results is provided below.

8.3.1 User Routine Tasks: Smart Hotel

We applied our approach in the automation of routines in the domain of a smart hotel. By applying our approach in this domain, we obtained routine tasks capable of adjusting their obtrusiveness level at runtime according to the user’s situational context. In this way, task automation behave in a considerate manner.

Specifically, we modeled a routine for automating the tasks when the

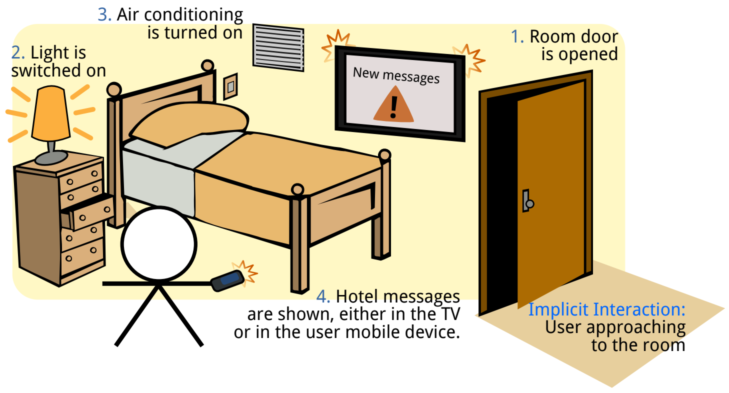

3. Air conditioning is turned on
2. Light is switched on
New messages
1. Room door is opened
4. Hotel messages are shown, either in the TV or in the user mobile device.
Implicit Interaction: User approaching to the room

Figure 8.10: Example scenario of a routine task adjusting the obtrusiveness level

user enters to his/her room. When a hotel guest approaches to his/her room (implicit interaction), a welcome hotel service will be automatically executed (see Figure 8.10): opening the door room to let the user come in, switching on the lights of the room, turning on the air conditioner to fit user preferences and communicating the new messages/news of the hotel to the user. Such tasks (which compose a routine) would be automatically performed just by approaching to the room (without the user explicitly requests them).

Applying our approach, the execution of each task behaves in a considerate manner, demanding the proper level of user attention in each situation. Considering the routine above introduced, each time it is performed, some of its actions could require a different degree of user attention. As a consequence, different interaction resources should be used to support each task at the appropriate obtrusiveness level (e.g., using either the TV or the mobile device to show the hotel messages).

Figure 8.11 shows the part of the interaction resources that supports some tasks from the "entering the room" routine. For example, the *ambient sound* of the room is used for the "communicate hotel messages" task when the task is performed proactively at the highest level of attention. However, the *mobile notification* and the *mobile vibration* are used for the same task when it is performed at the slightly level of

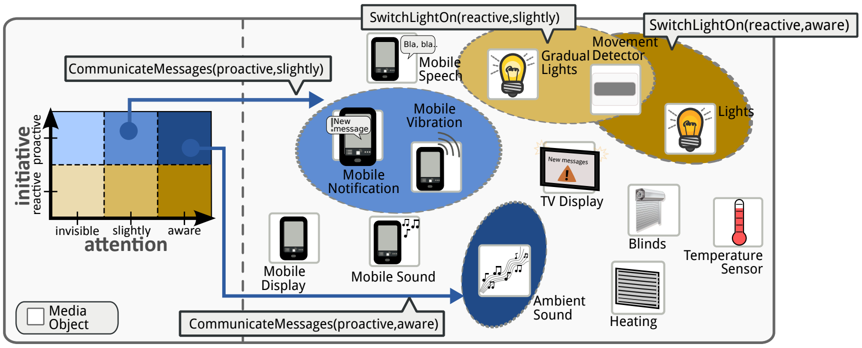


Figure 8.11: Implication of the interaction resources with the attentional demand

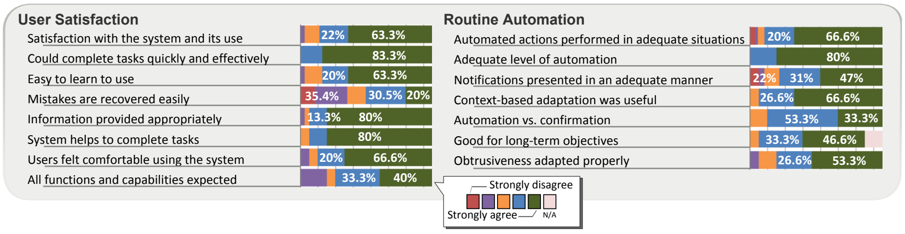


Figure 8.12: Summarized results

attention. Another example is the "switch light on" task. The *gradual lights* that regulate light's intensity are used for this task when it is performed at the slightly level of attention. Conversely, the *regular lights* are used when the task is performed at the highest level of attention (i.e., *completely-awareness*). Figure 8.11 illustrates all these mappings for the tasks.

The case study was defined following our design method and a prototype was developed to evaluate the user satisfaction with the execution of the routines. In particular, we wanted to evaluate whether the routines were automated adequately and whether guests were satisfied with the result system. Figure 8.12 shows a summarized table of the obtained results. Overall, the experiment showed that by following

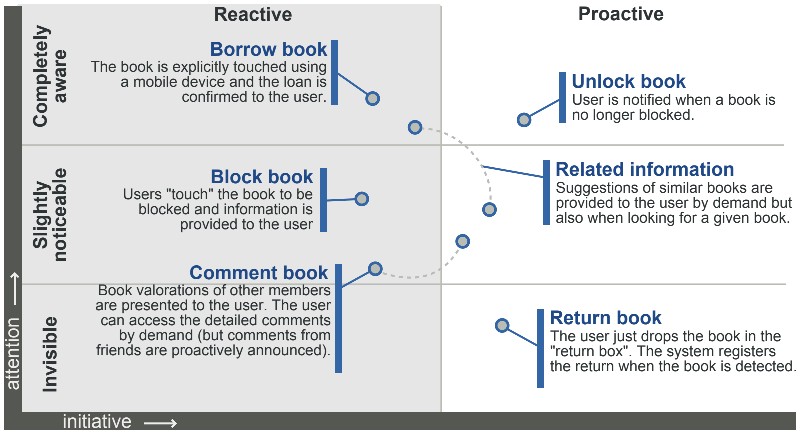

Reactive
Proactive
Completely aware
Slightly noticeable
Invisible
attention ⟶
initiative ⟶
Borrow book
The book is explicitly touched using a mobile device and the loan is confirmed to the user.
Unlock book
User is notified when a book is no longer blocked.
Block book
Users "touch" the book to be blocked and information is provided to the user
Related information
Suggestions of similar books are provided to the user by demand but also when looking for a given book.
Comment book
Book valorations of other members are presented to the user. The user can access the detailed comments by demand (but comments from friends are proactively announced).
Return book
The user just drops the book in the "return box". The system registers the return when the book is detected.

Figure 8.13: Obtrusiveness level defined for the services in the Smart Library scenario.

our method, user routines can be automated adjusting its obtrusiveness level according to context.

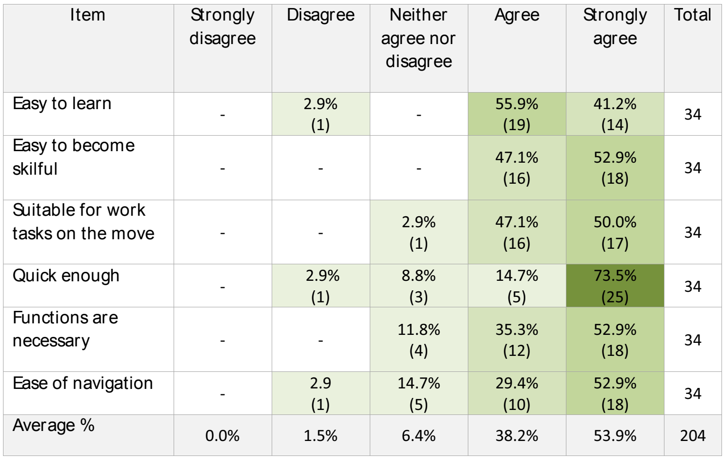
8.3.2 Smart Library

Once our proposal was established, we applied it for the development of Presto (Giner et al., 2010). Presto is a context-aware mobile platform that allows to support different workflows by interacting with the physical environment. Depending on the physical context, user interfaces were adapted for completing the workflow tasks. In particular, we applied our approach for the support of different processes that take place in a library such as borrowing a book, obtaining user comments and finding similar books. Adaptation in these scenarios takes into account different factors. For example, users are reminded for returning a book by means of a different interaction mechanism depending on the remaining time for their loan (first in a subtle manner, and later more energically), comments are displayed differently depending on the social relationship between the users, etc.

Figure 8.13 illustrates the linkage between different services supported by this case study and the unobtrusive adaptation space for the interactions that support each service. In the case study, we considered that it is important to inform users about the events that require some actions for them to be performed or events that are relevant for them. This is why services such as *borrow book* and *unlock book* were presented at the foreground of user attention (in a notorious manner). On the contrary, the *return of books* was performed in a completely unobtrusive manner, the user leaves the book in the return box and the system initiates the return process without notifying the user. In this way, users avoid queues. In the case of the *related information* and *comment book* services, not all the information that is associated with a book is relevant for the library users at anytime. So, we provided the interaction at a different obtrusiveness level depending on the relevance of the information. User comments that were made by friends were considered to be more relevant that those made by people that was not part of the user social network. Thus, comments from friends were provided in a proactive manner to the user. For *similar books*, books that were physically close to the user position were considered more relevant. Information of close books was provided in a proactive manner, but it was only shown as a hint (*slightly noticeable*).

In order to validate this case study, we applied it to support services in our faculty library. We developed a prototype version for the system and conducted an experiment. The experiment was performed by 34 users (26 men and 8 women, all between the ages of 20 and 60). The participants evaluated the prototype in different scenarios that comprised several alternate, simultaneous, and repetitive tasks playing the library user role. They were required to borrow books, make temporal reservations while looking for similar books with a better rating, and buy a copy of the book in an on-line store.

For the evaluation, we made use of the MoBiS-Q questionnaire (Vuolle et al., 2008). MoBiS-Q evaluates user experience combined with enhancements in work productivity. Questions related to the "*perceived usability of a mobile business service*" dimension are the most relevant for the current work since these questions are related with interaction. Figure 8.14 provides the detailed results for each question considered.



Item	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree	Total
Easy to learn	-	2.9% (1)	-	55.9% (19)	41.2% (14)	34
Easy to become skilful	-	-	-	47.1% (16)	52.9% (18)	34
Suitable for work tasks on the move	-	-	2.9% (1)	47.1% (16)	50.0% (17)	34
Quick enough	-	2.9% (1)	8.8% (3)	14.7% (5)	73.5% (25)	34
Functions are necessary	-	-	11.8% (4)	35.3% (12)	52.9% (18)	34
Ease of navigation	-	2.9 (1)	14.7% (5)	29.4% (10)	52.9% (18)	34
Average %	0.0%	1.5%	6.4%	38.2%	53.9%	204

Figure 8.14: Detailed results

Overall, the experiment showed positive results for the interaction aspects endorsing our proposal in this domain.

8.3.3 HomeCare

We applied our approach in a HomeCare case study to develop context-aware interactions adaptive to the different devices that the user possesses. To exemplify the proposal, we described a scenario of a HomeCare system that contain services to assist to users in their healthy habits.

The scenario described a day in John's life and the way interaction of the services changed depending on his context. John is a 60 years old man who lives in a smart home. He has a heart disease that forces him to take a pill at a certain time, but he is rather forgetful and he prefers to be reminded. He also appreciate if the house notify him when he has a doctor's appointment.

One day John was having breakfast while he was watching the TV and he forgot taking his pills, so a message was shown on his TV reminding him about it. When he took the pills, the system detected that

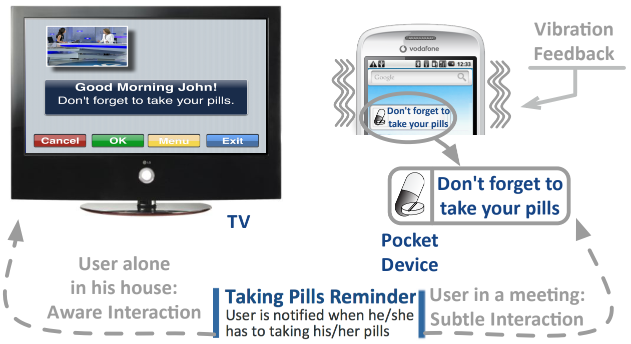

Good Morning John!
Don't forget to take your pills.
Cancel
OK
Menu
Exit
TV
Vibration Feedback
Don't forget to take your pills
Don't forget to take your pills
Pocket Device
User alone in his house: Aware Interaction
Taking Pills Reminder
User is notified when he/she has to taking his/her pills
User in a meeting: Subtle Interaction

Figure 8.15: Different interaction for the same service

there were few left and in reaction to this, the system added it to the medicine shopping list in an invisible manner for John. Later, when he was at his workplace, a message was shown on his mobile phone that reminded him about a doctor's appointment for a revision that afternoon. Finally, when he was having a business lunch at a restaurant, an alert was shown in his mobile phone that reminded him to take his pills. In this case, the alert was shown in a subtle way because John was on a meeting with other people. Figure 8.15 depicts the different ways that the taking the pills reminder are presented in the scenario.

Figure 8.16 shows the obtrusiveness modeling for the services that supported the case study.

In order to validate if the adaptation provided by different devices was satisfactory for the end-users, we developed a prototype version of the services described and conducted an experiment with users. The experimental setup included an HTC Magic mobile device running Android Operating System and a LG Scarlet TV using CE-HTML (simulated). Wizard of Oz techniques were used to simulate the adaptation process.

Figure 8.17 shows a summarized table of the obtained results. Overall, the initial results obtained showed that by following our approach, we can obtain user interfaces for multiple devices that support the specified services with the properly interaction mechanisms for each context

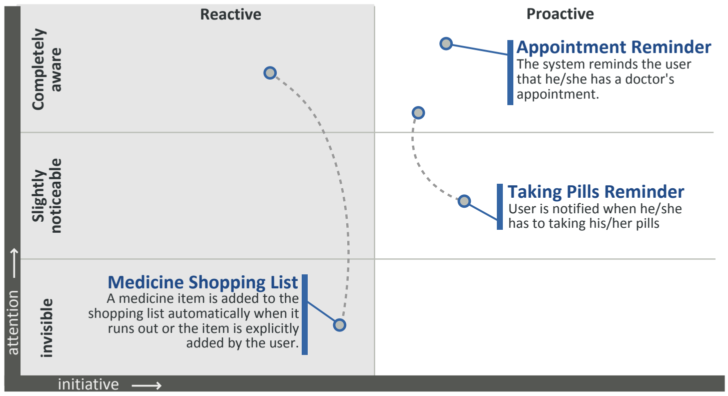

Reactive
Proactive
Completely aware
Slightly noticeable
invisible
attention ⟶
initiative ⟶
Appointment Reminder
The system reminds the user that he/she has a doctor's appointment.
Taking Pills Reminder
User is notified when he/she has to taking his/her pills
Medicine Shopping List
A medicine item is added to the shopping list automatically when it runs out or the item is explicitly added by the user.

Figure 8.16: Obtrusiveness level defined for the services in the SmartCare scenario.

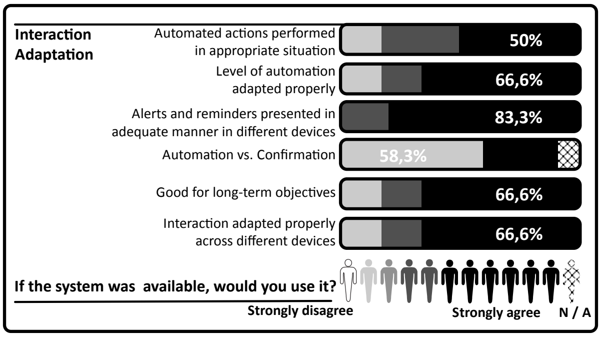


Figure 8.17: Summarized results

of use.

8.4 Discussion of the results

Based on our experiences from all the case studies, we present the lessons that we learned in the application of our approach.

8.4.1 Benefits obtained

The development of services' interactions obtrusiveness adaptations involves some aspects that are not usually considered in software development such as the interaction variability and the obtrusiveness constraints. The application of our development method has been useful in the development of this kind of systems at different levels. We have applied the approach in different domains. In this way, we could experience whether the approach was appropriate for supporting interaction obtrusiveness adaptation under different circumstances. The main benefits observed during the application of our approach are introduced below.

Error reduction

One of the conclusions from applying our method is that it is difficult for developers to keep an overall view of the system while focusing on a specific part of the adaptation. When we introduced our design method supported by tools, it was useful since it assured general consistency for the adaptive behavior. At design time, the presented method provides abstract models that allow the requirements related to adaptive services interaction behavior to be captured. Using these abstract models, the system can be designed by using concepts that are much less bound to the underlying implementation technology and are much closer to the problem domain (Paternò, 2003). Thus, designers can focus on the main concepts (the abstractions, such as user, obtrusiveness, interaction, etc.) without being confused by many low-level details (Paternò, 2003). This makes the provided models easier to specify, understand, and maintain than source code.

In addition, model-based validations ensure that the pieces defined fit together. At implementation, code generation is in charge of producing the code that is required for the adaptive behavior defined for interaction.

System evolution

One of the benefits of our approach is that the developed autonomic infrastructure directly interprets the models at runtime in order to adapt the services' interactions according to the user's attentional resources. Thus, the models become the primary means to understand, interact with, and modify the services interactions and their obtrusiveness level. This considerably facilitates the evolution of the system at runtime: as soon as the models are changed, the evolutions are applied by the autonomic infrastructure without the need to stop or redeploy the system.

8.4.2 Limitations detected

Since the approach has been applied in different domains, two complaints commonly found in adaptive systems were to some extent present in our approach. Specifically, to what extent users have control over adaptations and the handling of failures due to imperfect context sensing. 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, as the adaptation is transparently to users and automatically performed. This causes loss of control over the system and the feeling that the system is doing something "in our back". Context-aware applications should be *intelligible* (also called transparent, comprehensible, scrutable), capable of generating explanations of their behavior (Bellotti and Edwards, 2001). In this way, our system should explain its decisions to the user in some way that was not intrusive for him/her.

Handling failures

The designed adaptations are exposed to possible failures due to the imperfect context sensing or failures in the mobile device. As detailed in (Bellotti and Edwards, 2001), a context-aware system can not pretend to understand all of the user's context, thus it must be responsible about its limitations. The system should be able to handle these failures and self-heal accordingly at runtime. This can be achieved by means of defining reconfiguration rules to indicate the behavior of the system when a failure is detected.

8.5 Conclusions

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

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

9 Conclusions and Future Work

Where do we go?

I like the dreams of the future better than the history of the past
—Thomas Jefferson (1743-1826)

The present work has introduced a model-based approach to face the challenge of developing and adapting service interaction obtrusiveness in a mobile and ubiquitous context. Facing the development and runtime adaptation of such services' interactions in terms of obtrusiveness have resulted innovative and different contributions were produced from this work. In addition, the research line in which this work is aligned is by no means completed here. Further work can complement and extend this thesis.

This last chapter introduces the conclusions of the work developed in this thesis. First, Section 9.1 presents the main contributions to both the Context-Aware and Considerate communities. Section 9.2 provides an overview of the publications that have emerged from this work. Section 9.3 presents the projects co-directed related to some parts of this thesis. Finally, Section 9.4 outlines the ongoing and future work that can extend this research line, and Section 9.5 concludes with some final remarks.

9.1 Contributions

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

DSL for interaction obtrusiveness specification. Modeling primitives have been defined to facilitate the specification of the adaptive behavior of services' interactions according to the user's attentional resources. Separation of concerns and metamodeling techniques have been applied to **capture and organize formally the concepts** that conform this specification language.

Method for development. A systematic development method has been defined to guide the developer in the construction of considerate services adaptive to users' attention. The method comprises from specification to the final implementation. This method helps the different stakeholders in the design and development of this kind of systems **without duplicating efforts in the development**.

Runtime infrastructure for dynamic interaction adaptation. A self-regulating autonomic infrastructure has been defined in order to automatically adapt the interaction of the different services according to the momentary attentive state of users. **Design models are exploited at runtime** to drive the autonomic adaptation of interaction obtrusiveness.

System for learning. An obtrusiveness learning system has been defined in order to **improve the initial obtrusiveness design dynamically over time** according to the changing user preferences.

In this way, we maximize the user's satisfaction for a long-term use. Furthermore, the provided customization interfaces allow end-users to change their own obtrusiveness preferences manually **without the need to stop the system or redeploy it**. In this way, we also integrate the user in the self-adaptation loop.

To sum up, we do believe that using a model-driven approach is a promising way to self-adapt services' interactions in terms of obtrusiveness. As the whole approach is supported by models, feedback from users is easily mapped onto the models enabling future improvements at design time and at runtime. Also, the use of models to design unobtrusive interaction adaptations has been useful to centralize the knowledge about the services' behavior in a way that it is easy to handle for designers.

9.2 Publications

The research activity presented in this work has produced innovative and different contributions that have been presented and discussed on different peer-review forums. In this section, we present the articles in which this research has been published. Figure 9.1 outlines the publications achieved in the course of this thesis indicating the type of publication and the conference or journal where it was published. The order of the publications and the year when the publication was achieved is also illustrated by means of a path that starts from the problem awareness and ends with the present thesis. The bridge at the end of the path means that the last publication was submitted but it remains under the review process. Specifically in the figure, three types of publications are defined: journals, conferences, and workshops. The most relevant publications have a quality level associated, which corresponds to the JCR index for journals, and the CORE ranking for conferences. Also, conferences published in LNCS are indicated. The distinct publications in which the author of this thesis was involved are listed below.

International Journals Indexed in the JCR.

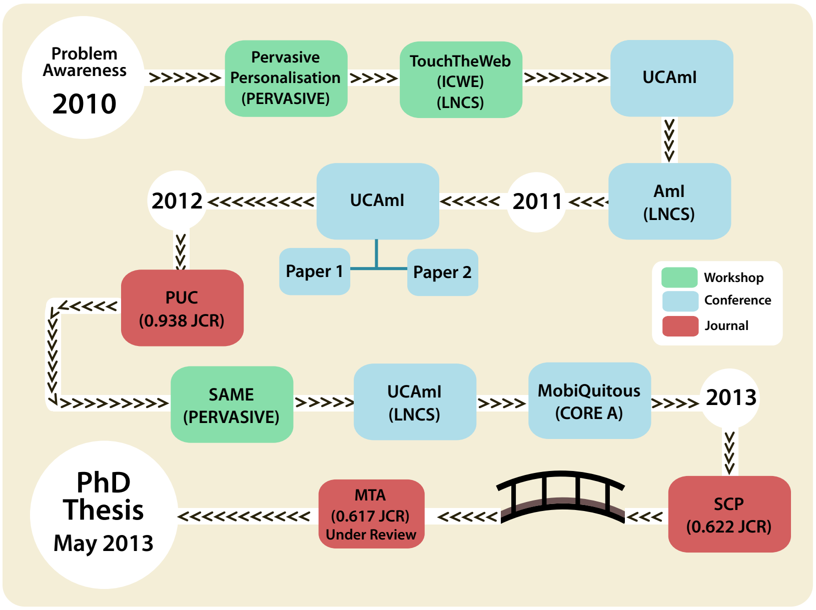


Figure 9.1: Publications overview.

1. **Miriam Gil**, Estefanía Serral, Pedro Valderas & Vicente Pelechano. *Designing for user attention: a method for supporting unobtrusive routine tasks.* Science of Computer Programming Journal, 2013. vol. 78(10), pp. 1987-2008.
2. **Miriam Gil**, Pau Giner & Vicente Pelechano. *Personalization for unobtrusive service interaction.* Personal and Ubiqui-tous Computing Journal, 2012. vol. 16(5), pp. 543-561. Springer.

International Conference Papers Indexed in the First Tier of CORE Ranking.

3. William Van Woensel, **Miriam Gil**, Sven Casteleyn, Estefanía Serral, & Vicente Pelechano. *Adapting the obtrusiveness of service interactions in dynamically discovered envi-*

ronments. 9th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous 2012).

International Conference or Workshop Papers published by Springer (LNCS).

4. **Miriam Gil** & Vicente Pelechano. *Exploiting User Feedback for Adapting Mobile Interaction Obtrusiveness.* 6th International Symposium of Ubiquitous Computing and Ambient Intelligence (UCAmI 2012), 2012. LNCS 7656, pp. 274-281. Springer, Heidelberg.
5. **Miriam Gil**, Pau Giner & Vicente Pelechano. *Service Obtrusiveness Adaptation.* First International Joint Conference on Ambient Intelligence (AmI 2010), Malaga, Spain, 2010. pp. 11-20. Springer Berlin / Heidelberg. Lecture Notes in Computer Science, Volume 6439.
6. Pablo Munoz, Pau Giner & **Miriam Gil**. *Designing Context-Aware Interactions for Task-Based Applications.* 10th International Conference on Web Engineering, ICWE 2010 Workshops, Vienna, Austria, 2010. pp. 463-473. Springer Berlin / Heidelberg. Lecture Notes in Computer Science, Volume 6385.

International Conference Papers.

7. **Miriam Gil**, Estefanía Serral, Pedro Valderas & Vicente Pelechano. *Achieving Unobtrusive Interaction for Routine Task Automation.* 5th International Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI 2011), 2011. pp. 37.
8. Ignacio Mansanet, **Miriam Gil**, Joan Fons & Vicente Pelechano. *A Feature-Based Approach For Context-Aware Interactions Over Multiple Devices.* 5th International Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI 2011), 2011. pp. 38.

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

International Workshop Papers published in High-Relevance Forums to this Thesis.

10. Estefanía Serral, **Miriam Gil**, Pedro Valderas & Vicente Pelecha-no. *Unobtrusive Personalized Services in Ambient Media Environments.* 5th International Workshop on Semantic Ambient Media Experience in conjunction with Pervasive 2012 (SAME 2012), 2012.
11. **Miriam Gil**, Pau Giner & Vicente Pelechano. *Service Obtrusiveness Personalisation.* Pervasive Personalisation Workshop held in conjunction with Pervasive 2010, Helsinki, Finland, 2010. pp. 18-25.

Moreover, we have one journal paper under review process:

- Estefanía Serral, **Miriam Gil**, Pedro Valderas & Vicente Pelecha-no. *Automating Unobtrusive Personalized Services in Ambient Media Environments.* Multimedia Tools and Applications Journal. MTA Special Issue on "Semantic Ambient Media – What has Been Achieved, and What still Needs to be Done". JCR 2011 impact factor: 0.617, position: 71/104 in the category "Computer Science/Software Engineering".

All these publications are exclusively associated with this thesis work. Table 9.1 shows the relation between the contributions presented in this thesis and the publications achieved during its execution.

9.2.1 Relevance of the publications

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

Contribution	Publication
DSL for specification	- SAME 2012 - AmI 2010 - UCAmI 2010 - Pervasive & ICWE Workshops 2010
Method for development	- SCP Journal 2013 - PUC Journal 2012 - UCAmI 2011
Runtime infrastructure	- SCP Journal 2013 - Mobiquitous 2012 - UCAmI 2011
Learning system	- UCAmI 2012 - Mobiquitous 2012

Table 9.1: Outline of the contributions and the publications achieved

Personal and Ubiquitous Computing. Personal and Ubiquitous Computing is a peer-reviewed scientific journal that has published some of the most innovative international research contributions on the design and evaluation of new generations of handheld and mobile information appliances. Since 1997, it has covered original research on ubiquitous and pervasive computing, ambient intelligence, and handheld, wearable and mobile information devices, with a focus on user experience and interaction design issues. According to the Journal Citation Reports (JCR), the journal has a 2011 impact factor of 0.938, ranking it 67th out of 135 journals in the category "Computer Science/Information Systems".

The article "*Personalization for unobtrusive service interaction*" published in this journal is part of an special issue entitled "Adaptation and Personalization for Ubiquitous Computing" which is focused on topics that are central to this thesis. The guest editors for this special issue were Zhiwen Yu, Doreen Cheng, Ismail Khalil, Judy Kay, Dominikus Heckmann, relevant researchers in the Ubiquitous Computing area.

Science of Computer Programming. Science of Computer Programming is a peer-reviewed scientific journal dedicated to the distribution of research results in the areas of software systems development, use and maintenance, including the software aspects of hardware design. According to the Journal Citation Reports (JCR), the journal has a 2011 impact factor of 0.622, ranking it 70th out of 104 journals in the category "Computer Science/Software Engineering".

The article "*Designing for user attention: a method for supporting unobtrusive routine tasks*" published in this journal is part of an special issue entitled "Software Engineering Aspects of Ubiquitous Computing and Ambient Intelligence" which is focused on techniques for modeling ubiquitous systems and leveraging them at runtime.

MobiQuitous. The International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services provides a forum for practitioners and researchers to interact and exchange experiences about the design and implementation of mobile and ubiquitous systems. According to the CORE conference ranking, MobiQuitous is a Core A conference.

The paper " *Adapting the obtrusiveness of service interactions in dynamically discovered environments*" was presented in the main track of this conference.

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

Ubiquitous Computing and Ambient Intelligence. UCAmI has been consolidated as a reference event in Ubiquitous Computing & Ambient Intelligence, agglutinating high quality papers. This conference provides a discussion forum where researchers and practitioners on Ubiquitous Computing and Ambient Intelligence can meet, disseminate and exchange ideas and problems, identify some of the key issues related to these topics, and explore together possible solutions and future works. The conference papers are published by Springer (LNCS).

International workshops. In addition to the above mentioned venues, different parts of the work have been published in workshops from relevant conferences such as Pervasive or ICWE. This has helped to achieve diffusion of the work.

9.3 Projects codirected

In addition, two degree projects and a master thesis were co-directed in the context of this work to explore some concepts and put into practice its application. In particular, the degree projects and the master thesis in which this work has been applied are the following:

- *Desarrollo de una aplicación móvil para la gestión de tareas personales.* Sergio Segarra Miró. September 2011. Degree project.
- *Personalización de perfiles de notificación de aplicaciones según el contexto del usuario.* Nacho López Guerrero. September 2012. Degree project.
- *Reconfiguración de la interacción en sistemas Android: adaptando las notificaciones al contexto.* Deisson Leonardo Sánchez Toledo. September 2012. Master Thesis.

Furthermore, three master thesis are being co-directed at the moment where other parts of the present work are being applied in practice.

9.4 Future work

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

Control over adaptations. Users using adaptive systems may not understand how these applications make their decisions, letting them alone to be aware when the decisions are made and actions are taken. This is what Bellotti and Edwards named the *intelligibility* (Bellotti and Edwards, 2001):

> "*Context-aware systems that seek to act upon what they infer about the context must be able to represent to their users what they know, how they know it, and what they are doing about it.*"

Offering to users the possibility to check the adaptations could reduce the uncertainty and lack of control over the adaptive system. Thus, future extensions can further explore these aspects by increasing the intelligibility of our adaptive system.

Adaptations in dynamic environments. In the present work, the adaptive behavior of the services' interaction has been considered within an environment with ubiquitous services and context conditions known a-priori. But, in the "open world" the user can enter to new and undiscovered smart environments, and interactions with the services of the new environments should also behave in a considerate manner. Thus, we propose the development of mechanisms capable of discovering these new environments and the services in there and adapt the interaction with these services in a considerate manner.

User behavior patterns. User behavior has been considered in the present work at the learning system level in order to improve the initial design according to the user's reaction. However, individual actions of users have been considered for this work. Studying and detecting user behavior patterns can be convenient in order to better adapt the system to the new user needs and preferences. Thus, more work is needed in this direction to understand user behavior and behavior patterns that can affect user preferences.

9.5 Final remarks

Paul Hemp, an editor of the *Harvard Business Review*, spent a week training to be a room-service waiter at a Ritz Carlton hotel in Boston. The training emphasized empathy with the guests and anticipation of their needs. The care and concern shown by the employees had to be genuine: *the staff really has to care about their guests. Every morning, they get together to discuss the day's guests and their desires. They review the service philosophy of the Ritz every day.* Their philosophy is:

> "If you go to a good hotel and ask for something, you get it... If you go to a great hotel, you don't even have to ask." The Ritz-Carlton Hotel chain's philosophy—they want to be great hotels.
> *—Paul Hemp (2002)*

This means that a service is often an experience and it has to be designed "to make users feel good so they use it again". Thus, services do not have to overwhelm users, at the contrary, they have to facilitate users life and anticipate to their needs. This thesis is an attempt to achieve this by providing users with services that behave like considerate humans.

A
Metamodels & Tool Support

The design method proposed in Chapter 5 defines a set of concepts that are used to describe the interaction obtrusiveness adaptation of ubiquitous services at different abstraction levels. Designers can combine these concepts to specify the services' obtrusiveness requirements for each persona, the interaction variability in terms of obtrusiveness, and the interaction components of the target technology according to the interaction features. A well-defined language must be used for this specification to avoid ambiguities.

This section formalizes the concepts used for describing considerate adaptations of services' interactions into a *Domain Specific Language* (DSL). A DSL is a focussed, processable language for describing concisely a specific concern when building a system in a specific domain (Voelter, 2013). Also, it provides abstractions and notations tailored specifically to that domain helping designers in the system specification.

The DSL defined for the specification of the considerate services' interaction behavior according to user's situations is named AdaptIO. AdaptIO describes in a formal manner the concepts involved in the design method and the specific ways in which these concepts can be combined to create a valid definition. Since AdaptIO has been defined to be machine-processable, the descriptions made using this language can be automatically handled by different tools. In particular, we have defined an **AdaptIO modeling suite** with the different tools to edit, validate and guide the design of this kind of services' adaptations.

The following subsections provide detail on the definition of the AdaptIO DSL and the support for validating the different aspects of the specifications defined with the language.

A.1 The AdaptIO modeling language

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

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

Specifically, the AdaptIO modeling language is composed by several models that conforms to its metamodels. It also integrates the Feature Model metamodel in a non-intrusive manner. Since the Feature Model metamodel is used as-is, we can make use of the existing tools for feature model definition.

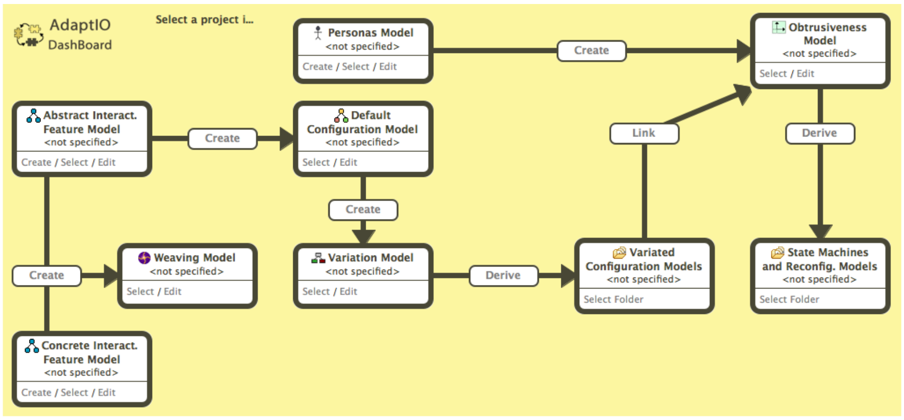


Figure A.1: AdaptIO dashboard

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

Figure A.1 shows the AdaptIO dashboard view that we provide to assist designers work through the flow of designing considerate interaction adaptations. As the figure shows, it invokes actions for many of the steps of our design method. Each action opens a graphical editor to create or edit the corresponding models. The following subsections give further details on these editors and the metamodels they conform to.

It is important to note that using this tool, the proposed models not

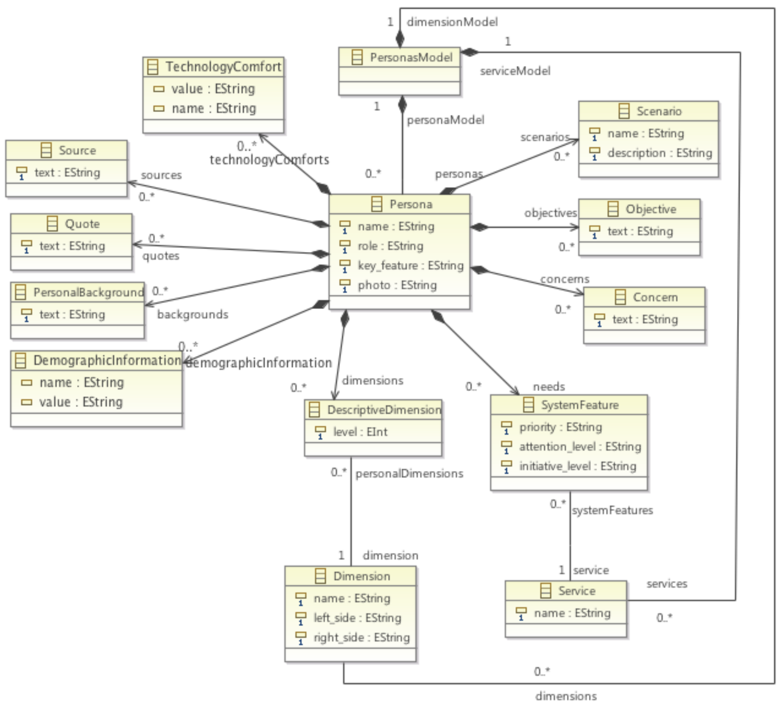
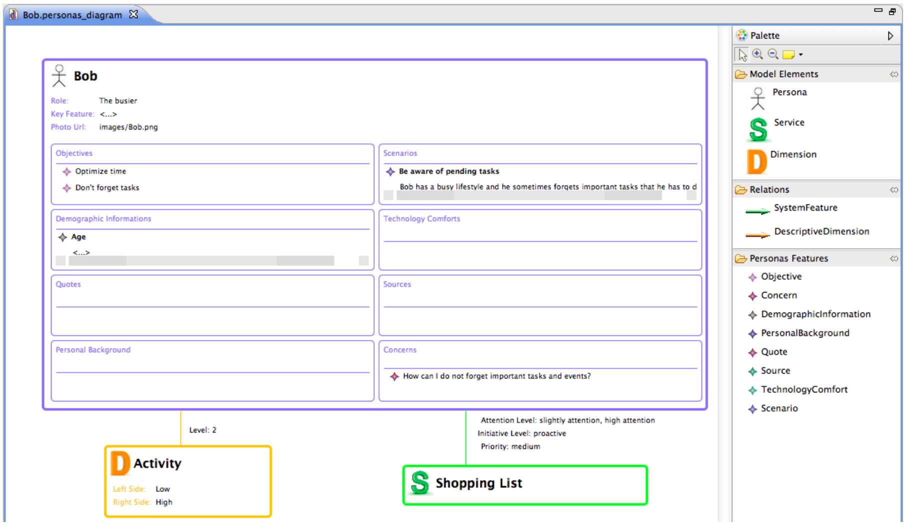


Figure A.2: Persona model metamodel

only can be graphically visualized and edited, but also, they are stored in XMI (XML Metadata Interchange), which is a machine-processable language. In addition, these tools also provide model-based validations to ensure that the specified models are valid.

A.1.1 The persona model metamodel

The persona metamodel defines in a formal manner the modeling concepts to design personas and specify their required services and obtrusiveness requirements. Figure A.2 shows the metamodel. The *PersonasModel* is composed by *Personas*, *Services*, and different *Dimen-*



Bob
Role: The busier
Key Feature: <...>
Photo Url: images/Bob.png

Objectives
Optimize time
Don't forget tasks

Scenarios
Be aware of pending tasks
Bob has a busy lifestyle and he sometimes forgets important tasks that he has to d

Demographic Informations
Age
<...>

Technology Comforts

Quotes

Sources

Personal Background

Concerns
How can I do not forget important tasks and events?

Level: 2

Activity
Left Side: Low
Right Side: High

Attention Level: slightly attention, high attention
Initiative Level: proactive
Priority: medium

Shopping List

Figure A.3: Persona editor

sions to characterize each persona. For each *Persona*, its relevant information can be defined such as *PersonalBackground*, *DemographicInformation*, *Quotes*, *Sources*, *TechnologyComfort*, *Scenarios*, *Objectives*, *Concerns*, *DescriptiveDimensions* and *SystemFeatures*. These *SystemFeatures* specify the services required for the personas to accomplish their goals and how these services are provided in terms of obtrusiveness.

The provided graphical editor is shown in Figure A.3. This editor supports the *obtrusiveness requirements definition* step of our design method (step 1). Using the dashboard view's "Create" or "Edit" links in the Personas Model section (see Fig. A.1), the editor is opened to create a new Persona model or edit an existing one. The "Select" link allow to locate an existing Persona model.

A.1.2 The obtrusiveness model metamodel

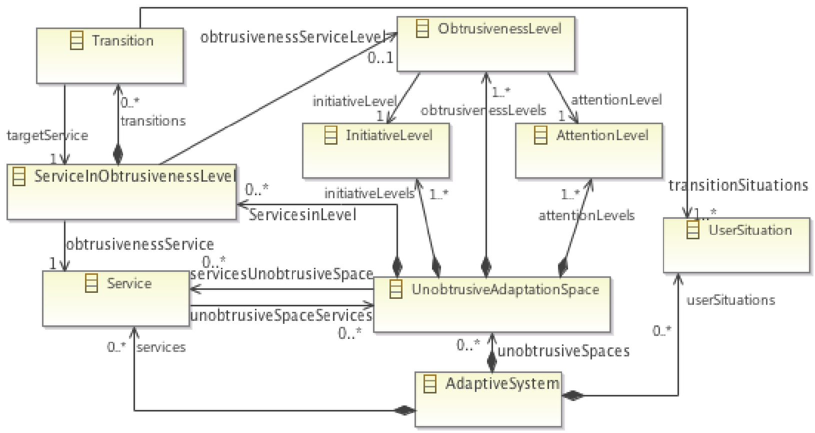


Figure A.4: Obtrusiveness model metamodel

The obtrusiveness model metamodel defines the concepts used for describing the obtrusiveness level of services for each persona. Figure A.4 shows an excerpt of the metamodel. As the figure illustrates, we represent the entire *adaptive system* by zero to many *unobtrusive adaptation spaces*. In this way, each unobtrusive adaptation space can define analogous interactions, since services in the same space are likely to face similar problems and can share solutions. This is useful to establish priorities of services, since each unobtrusive adaptation space can represent a different priority, and therefore, different adaptations.

An *unobtrusive adaptation space* is composed by *obtrusiveness levels* which describe the different possibilities in which interaction with the system can be offered. Each obtrusiveness level is formed by one *initiative level* to indicate who initiates the interaction (e.g., the user or the system) and one *attention level* to indicate to which degree the interaction intrudes on user's mind. Each *service* in the system is located in the possible obtrusiveness levels in which the service can be performed. Services in the obtrusiveness levels have *transitions* that specify how a particular service should move between obtrusiveness levels to adapt its interaction according to *user situations* (inferred from context changes expressed as rules).

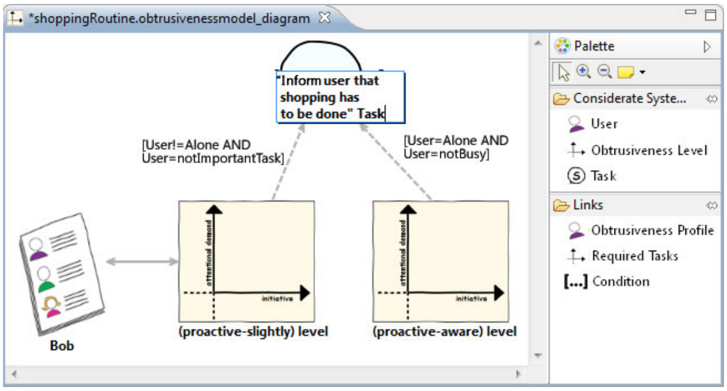

*shoppingRoutine.obtrusivenessmodel_diagram
"Inform user that shopping has to be done" Task
[User!=Alone AND User=notImportantTask]
[User=Alone AND User=notBusy]
Bob
(proactive-slightly) level
(proactive-aware) level
attentional demand
initiative
Palette
Considerate Syste...
User
Obtrusiveness Level
Task
Links
Obtrusiveness Profile
Required Tasks
[...] Condition

Figure A.5: Snapshot of the obtrusiveness modeling tool

Figure A.5 shows the graphical tool that support the definition of the obtrusiveness model. Following the dashboard, when designers select the “Create” link between the Personas Model and the Obtrusiveness Model, the obtrusiveness model is created and initialized based on the information of the personas model. In this way, designers only have to complete the model.

A.1.3 The feature model metamodel

In order to model the interaction variability and the concrete interaction, we have used the Feature Model provided by Moskitt4SPL[1]. Moskitt4SPL is a free open-source tool that is part of the Moskitt modeling suite[2]. Moskitt4SPL is defined as a set of plug-ins that we could incorporate to enhance our tool support with software product lines (SPL) capabilities.

Moskitt4SPL provides features that are well suited for the use we are making of Feature Models. Figure A.6 shows the different concepts in the Feature Model metamodel and the relationships among them. The metaclass *FeatureModel* normally is used as the root element of

[1]http://www.pros.upv.es/m4spl
[2]http://www.moskitt.org

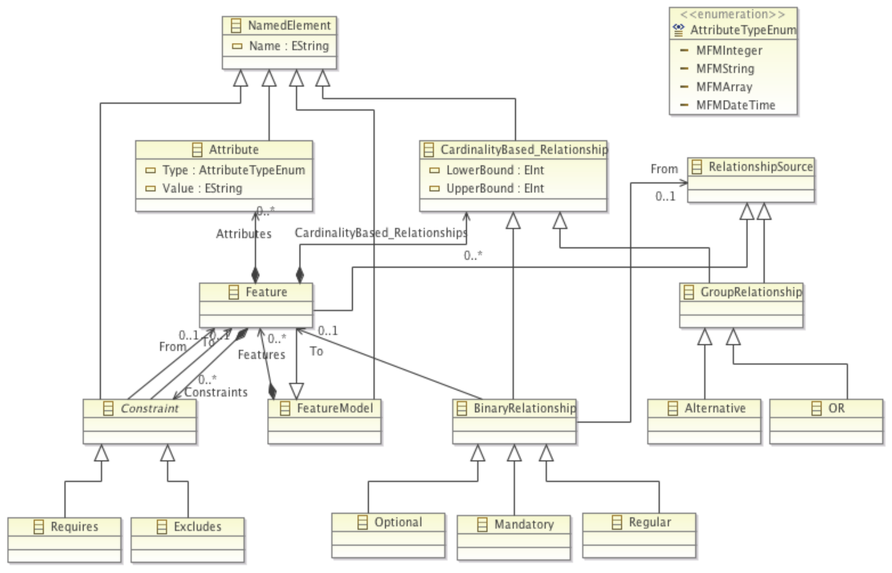


Figure A.6: Feature Model metamodel

Feature models. The *Feature* metaelement represents the different features of the Feature model. Each feature can have attributes that are represented by the *Attribute* metaelement. Features are related among them through relationships represented by the *CardinalityBasedRelationship* metaelement. This metaelement is specialized in the different relationships that the Feature model supports: *Or*, *Alternative*, *Optional*, *Mandatory*, and *Regular*. Also, the *Constraint* metaelement represents the constrants that can exist between features: *Requires* and *Excludes*.

This metamodel is based on the generic formalization of the Feature Model syntax defined by Schobbens et al. (Schobbens et al., 2007). According with the results of their work, the feature model editor incorporates support to multiple graphical notations. Users can dynamically **change the graphic notation** of feature models. This is very convenient when dealing with large user interface models (see Fig. A.7).

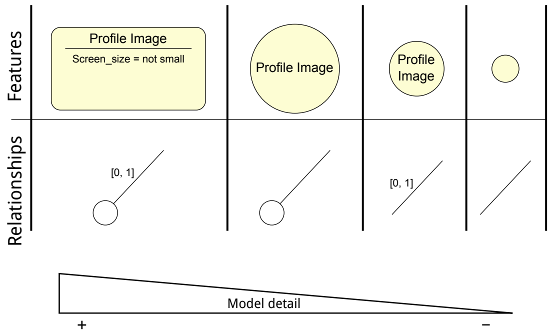


Figure A.7: Different representations for interface nodes

The Feature mModel editor supports customizing the notation at any time between the following feature representations:

1. **Feature with Attributes.** Features are graphically represented by means of rectangles. These rectangles are composed of two compartments. The top compartment holds the feature name and the bottom compartment holds the features attributes. These features attributes follows the pattern: <name>=<value>.
2. **Rounded Feature.** Features are graphically represented by means of ellipses. The feature name is at the ellipse center, whereas feature attributes are not shown.
3. **Fixed Feature.** Features are represented as Rounded features which diameter depends of the feature name length.
4. **Simplified Feature.** Features are graphically represented by means of ellipses. Neither the feature name is visible, nor the feature attributes. The ellipse diameter is set to a constant.

It also supports customizing relationship notation as follows:

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

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

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

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

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

Figure A.8 shows the environment of the Feature Model editor. This editor is opened from the dashboard by selecting the "Create" or "Edit" links in both the *Abstract Interact. Feature Model* section and the *Concrete Interact. Feature Model* section. This editor is used to create the interaction variability model and the concrete interaction model. To model the concrete interaction model, we use the *Regular* relationship of the feature model editor, which express a simple containment relationship. Figure A.9 shows the way to change between the different visualizations supported from the Feature Model editor.

Then, these models are mapped by means of a Weaving Model. In order to create the weaving model, designers have to select the "Create" link between both models. Figure A.10 shows a snapshot of the weaving model editor using ATLAS Model Weaving tool (Fabro et al., 2006). The interaction features appears in column 1 and Atlas Model weaving (column 2) establishes the relations between features and concrete

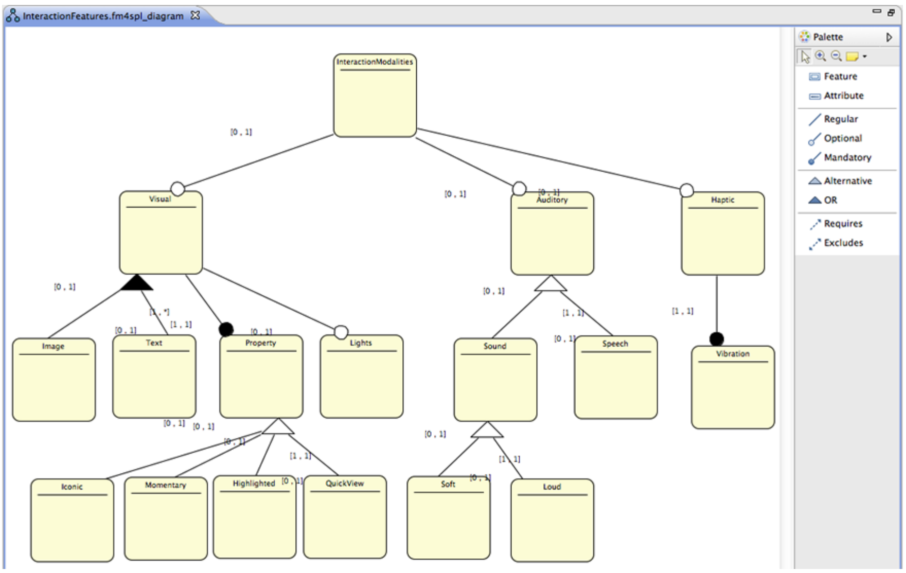


Figure A.8: Feature model editor

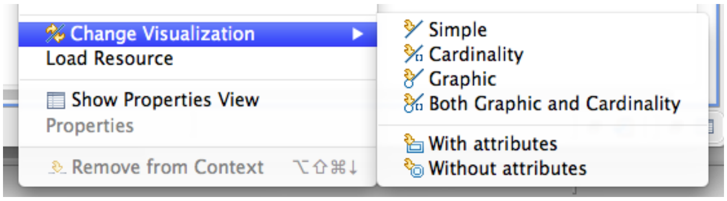


Figure A.9: Changing the visualization mode from the editor

interaction components (column 3).

Using the "Create" link between the *Abstract Interact. Feature Model* and the *Default Configuration Model*, the *Feature Model Configurator* is opened to create a default configuration from the defined Feature Model. A configuration of a Feature Model conforms to the metamodel of Figure A.11. In this metamodel the *ConfigurationModel* metaelement is composed by the set of *FeatureStates* of a Feature Model.

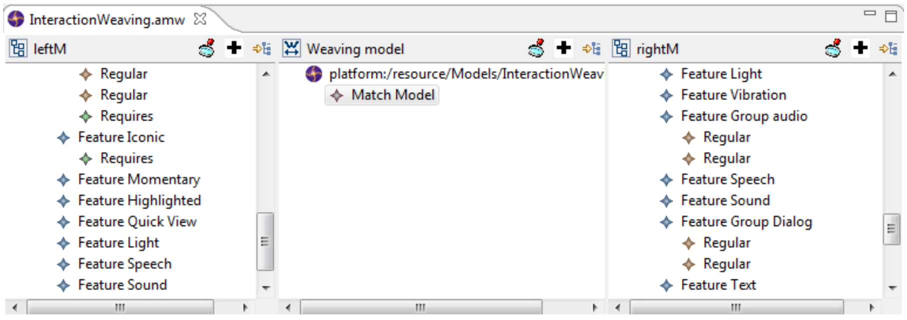


Figure A.10: Snapshot of a Weaving Model.

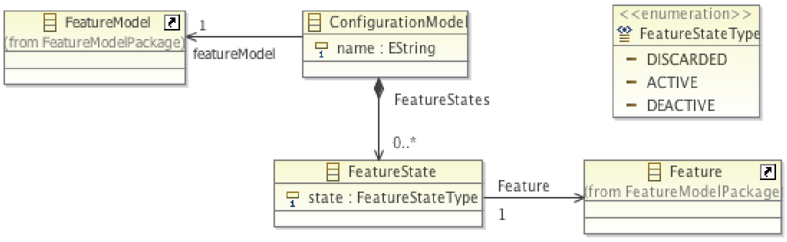


Figure A.11: Configuration model metamodel

The feasible feature states are: active, deactive or discarded. The editor that supports the Feature Model Configurator is shown in Figure A.12.

From the default configuration model, designers can define the interaction variability according to each obtrusiveness level. To do this, they click on the "Create" link and the default configuration model is loaded allowing to express increments and decrements of features with respect to this configuration. Figure A.13 shows the main window to create variations. From this window, designers can create new variations by simply clicking the "Add Variation" button. This button opens the variation editor that allows designers to express the variations from a configuration model (see Fig. A.14).

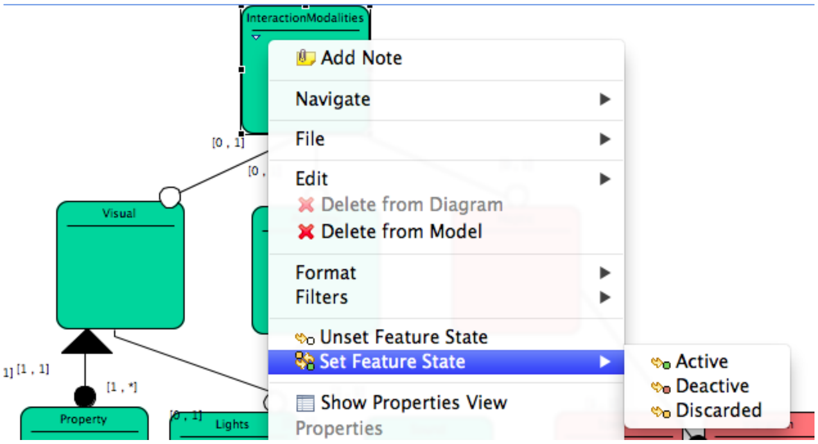
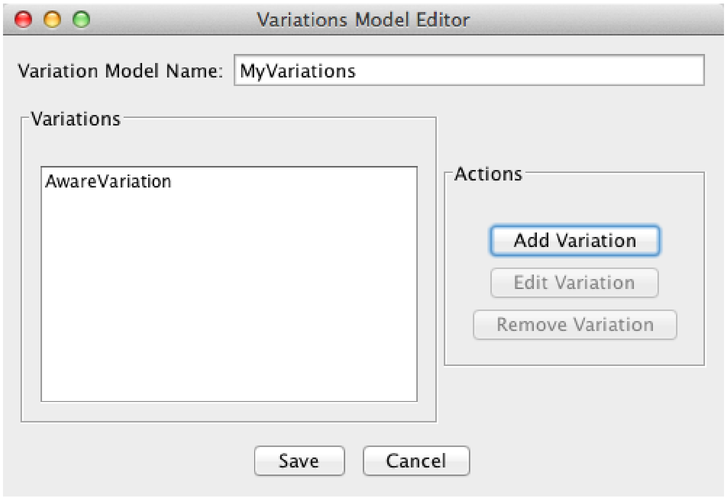


Figure A.12: Feature model configurator editor


Variations Model Editor

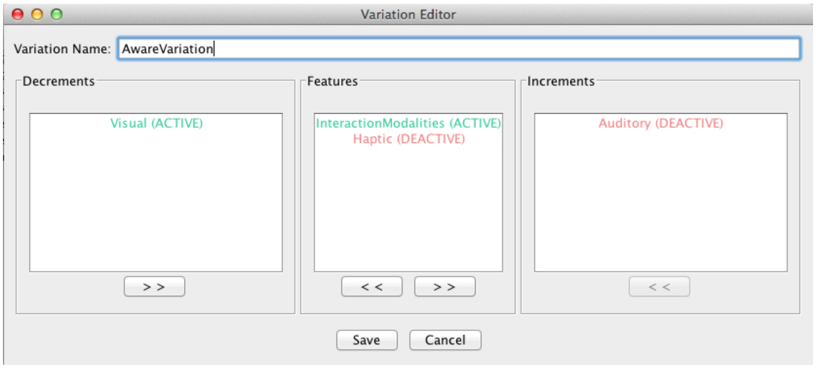
Variation Model Name: MyVariations

Variations

AwareVariation

Figure A.13: Variations Model editor

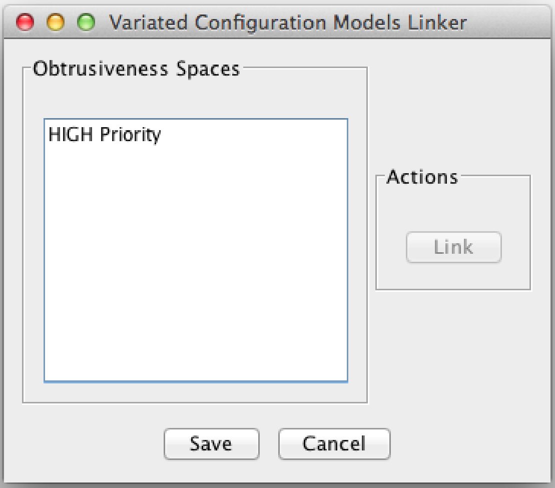
Then, using the "Derive" link between the *Variation Model* and the *Variated Configuration Models*, the different configuration models are derived automatically. These configuration models are used to specify



AwareVariation

Decrements	Features	Increments
Visual (ACTIVE)	InteractionModalities (ACTIVE)	Auditory (DEACTIVE)
	Haptic (DEACTIVE)	

Figure A.14: Variations editor



HIGH Priority

Figure A.15: Editor to link interaction configurations to the unobtrusive adaptation spaces

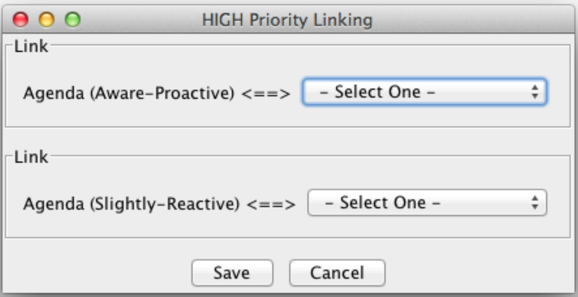


Figure A.16: Editor to link interaction configurations to a service in an obtrusiveness level

the interaction configuration for each obtrusiveness level. Thus, using the "Link" link between the *Variated Configuration Models* and the *Obtrusiveness Model* these associations are done graphically. Figure A.15 shows the main window where designers can choose the unobtrusive adaptation space to do the mapping. When they select an unobtrusive adaptation space, a list of the services in that space is shown and designers can select one configuration to do the association (see Figure A.16).

Once these models are created and traced, designers can select the "Derive" link of the *Obtrusiveness Model* in order to obtain the state machines that define the adaptive behavior of the different services. These state machines capture the implicit adaptive behavior of the services' interactions according to user situations. They are used to adapt the service interactions at runtime. The way to leverage these models at runtime is explained in Chapter 6.

A.1.4 The context model metamodel

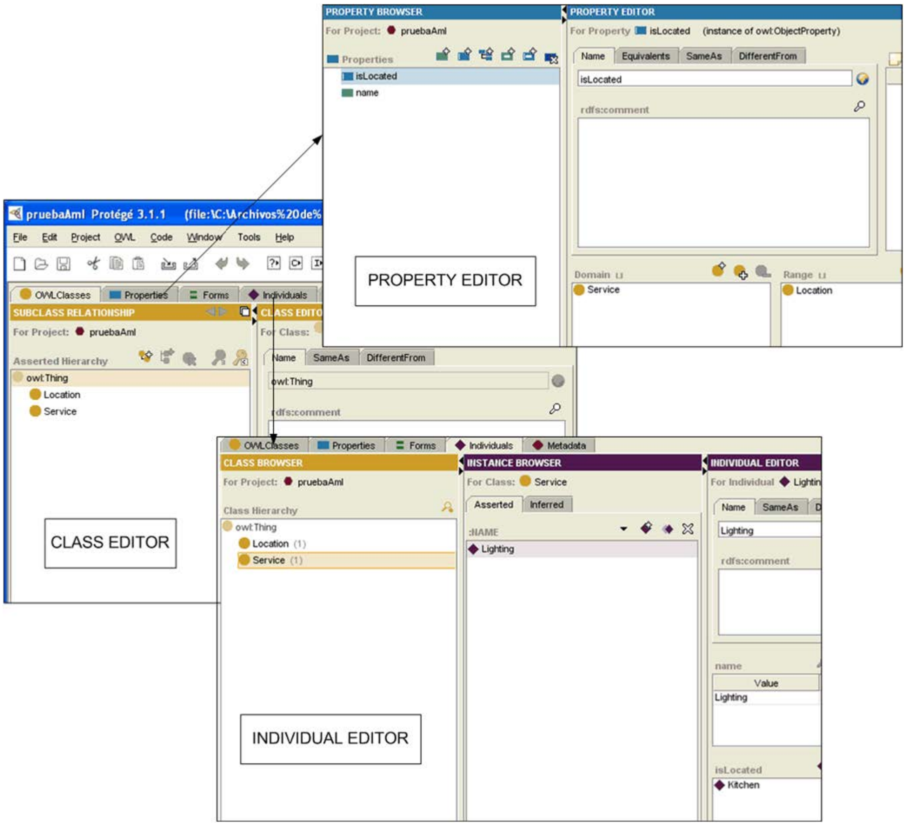


Figure A.17: Snapshop of the Protégé user interface

There are several tools that can be used to create OWL models. Some examples are SWOOP[3], the editor developed by the Model Feature company[4], the OWL Visual Editor of the EMF Ontology Definition Metamodel (EODM)[5] plugin developed upon the eclipse platform, the

[3]http://www.mindswap.org/2004/SWOOP/
[4]http://www.modelfutures.com/
[5]http://wiki.eclipse.org/MDT-EODM

SematicWorks tool developed by Altova[6] or Protégé[7]. Any of these tools can be used to create the OWL context model. We have used the Protégé tool because it is an open source tool that can be freely downloaded from its Web page, and because it provides a very intuitive interface to create ontologies. In addition, there is a great research community that is continuously extending and improving this tool. A proof of that is the International Protégé Conference that is celebrated every year.

Figure A.17 shows a snapshot of the Protégé tool. In this snapshot the above introduced OWL model is being created. The Protégé user interface is divided in several tabs that provide us with editors to create the different elements of the ontology: classes, properties and individuals.

A.1.5 The Android components metamodel

The Android components metamodel defines in a formal manner the modeling concepts of the Android application framework. The metamodel is shown in Fig. A.18. The metaclass *App* is the root element of the component architecture model. The metamodel includes the definition of the Android components (*Activity*, *Service*, *ContentProvider*, and *BroadcastReceiver*) and the communication mechanisms among them (*LaunchImplicit*, *LaunchExplicit*, *IntentFilter*, and *IntentBroadcast*).

A graphical editor was provided for describing the components of different Android-based applications. As Fig. A.19 shows, the developed tool incorporates a palette of Android components that can be labeled and linked with other components. The components defined are the ones introduced in Chapter 6.

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

[6]http://www.altova.com/products/semanticworks/semantic web rdf owl editor.html

[7]http://protege.stanford.edu/overview/protege-owl.html

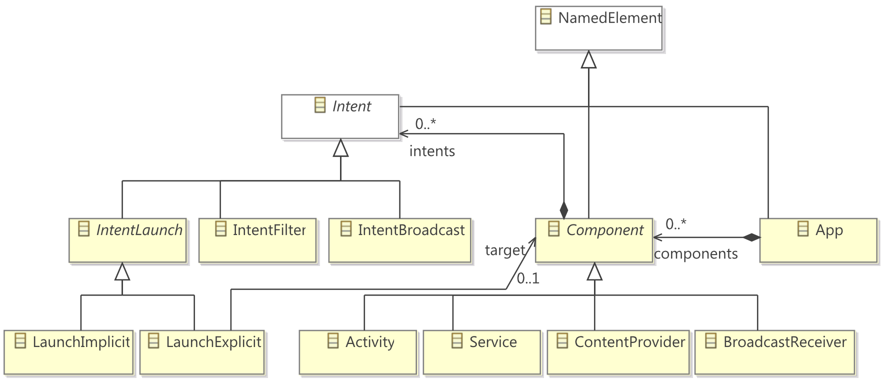


Figure A.18: Android components metamodel

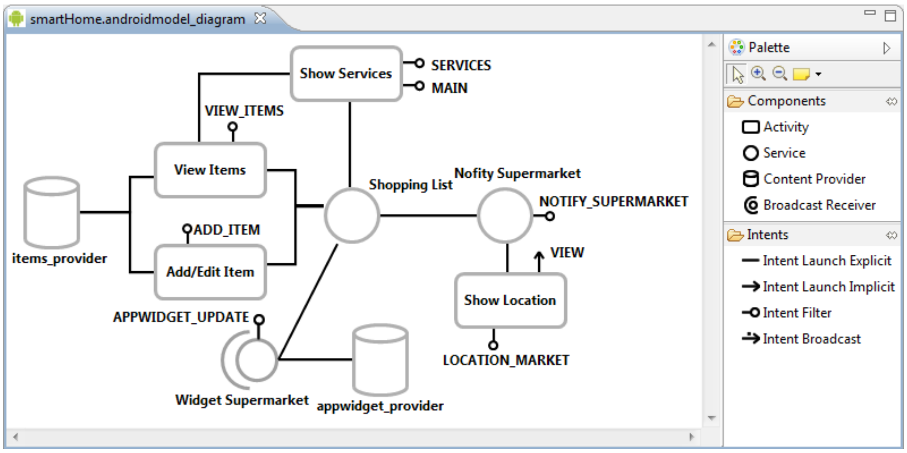


Figure A.19: Graphical editor for Android components

ones.

A.1.6 Relationships between metamodels

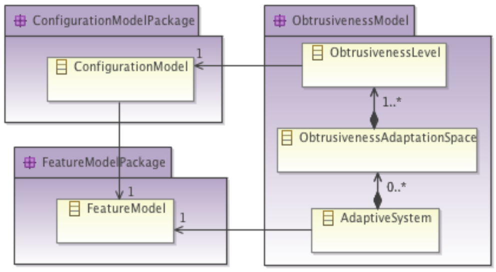


Figure A.20: Dependency relationships between metamodels

In order to match the appropriate interaction configuration to each service in an obtrusiveness level, we link the elements of the different models. In this way, when a change in the obtrusiveness level for a service is produced, the interaction is adapted according to the new configuration. These links are represented by means of dependency relationships between the metaelements of the different metamodels in the following manner (see Fig. A.20):

- The *AdaptiveSystem* in the obtrusiveness model metamodel is related with a unique *FeatureModel* in the feature model metamodel, since the *FeatureModel* represents all the available interaction features of the system and their relationships.

- Each *ObtrusivenessLevel* in an unobtrusive adaptation space has associated a *ConfigurationModel* from the configuration model metamodel. This represents the interaction configuration assigned to that obtrusiveness level.

Note that these dependency relationships do not automatically create any references; it merely grants permission for them to be established.

A.2 Model-based validation

One of the most important uses of models is to reason about the system they describe. Model-based verification can ensure that some aspects of the system are valid prior to its construction. This section introduces some of the validation capabilities provided during system specification.

The method proposed for the design of service interactions obtrusiveness adaptation promotes separation of concerns. Different aspects are specified in different models in order to better handle complexity. However, this requires to apply validation techniques in order to guarantee that the different aspects specified are consistent.

We have implemented different validations that can be applied automatically to the AdaptIO models. The validation mechanisms are supported by two different ways. First, we have expressed constraints in the different models to check the properties that have to be fulfilled by the models. Second, we have used analysis tools on Feature Models to check the validity of the Feature Model and interaction configurations defined.

A.2.1 Constraints on models

These validations are specified at the metamodel level. This allows designers to automatically validate a created model from the metamodel. Specificaly, they check on different constraints expressed in the metamodel. An overview of the questions that can be answered thanks to these contraints is provided below.

Are the different elements on the model unique? All the names of the different elements of a model (e.g., personas, services, user situation, obtrusiveness space, etc.) need to be unique since they are the identifiers of the element.

Are all the obtrusiveness levels supported by an interaction configuration? Given an unobtrusive adaptation space, all the obtrusiveness levels that compose it need to be linked to an interaction configuration of the feature model.

Is a service interface suitable for generation in a given user

situation? Given a specific user situation, the traceability between all the models (from a persona to the concrete interaction components model) have to guarantee a valid generation of a service interface for that user situation. In this way, we verify that no problem is produced when interaction components are generated.

The constraints have a twofold goal. On the one hand, they are used during design to provide designers with immediate feedback about inconsistencies in their specifications. On the other hand, they simplify the definition of the model transformations since transformations can be specified by assuming that the models are consistent.

A.2.2 Reasoning on feature models with FAMA

In order to validate the Feature Model description and the different variation configurations, we use the *Feature Model Analyser Framework* (Benavides et al., 2005) (FAMA). This framework implements the automated analysis of Feature Models using Constraint Satisfaction Problems (Jaffar and Maher, 1994). Figure A.21 shows the FAMA operations available for a Feature Model. In particular, we use the following operations.

Validate model. This operation checks if a feature model is not empty or has at least one interaction configuration.

Validate product. This operation determines if an interaction configuration is valid for a given feature model.

Detect errors on model. This operation looks for errors on a feature model.

Explain errors on model. When a feature model has errors, this operation looks for explanations (relationships) for the errors.

Explain error on product. This operation provides options to repair an invalid interaction configuration for a given feature model.

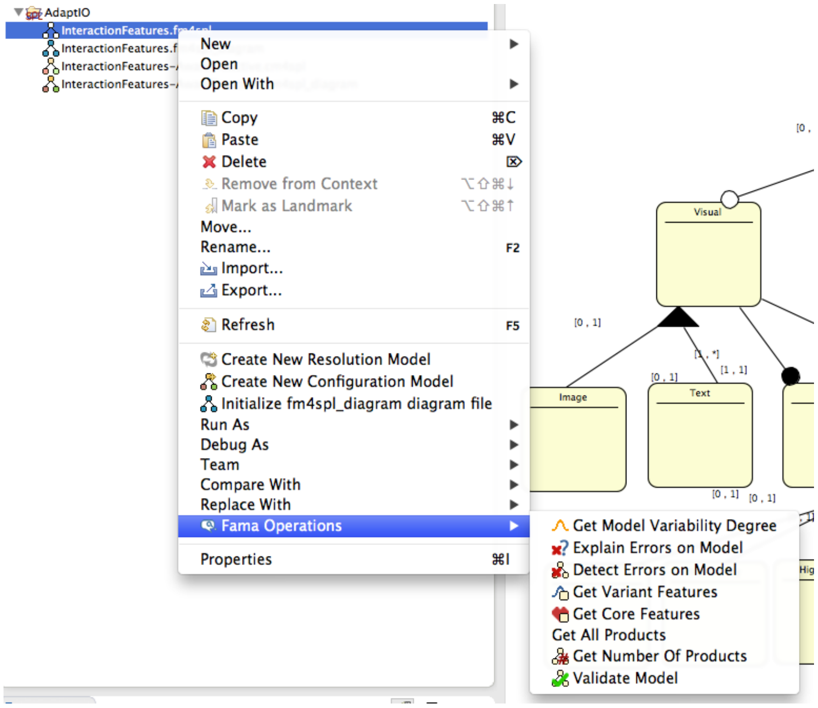
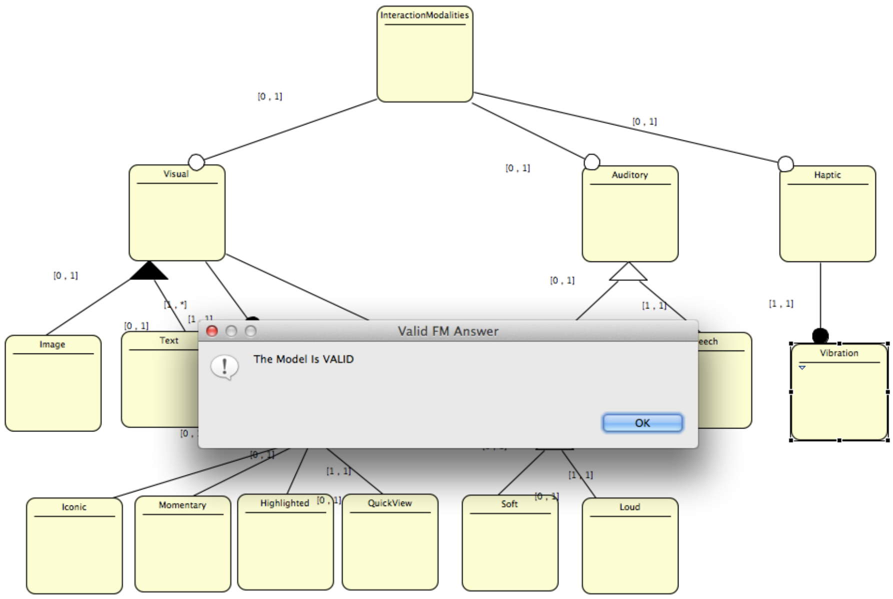


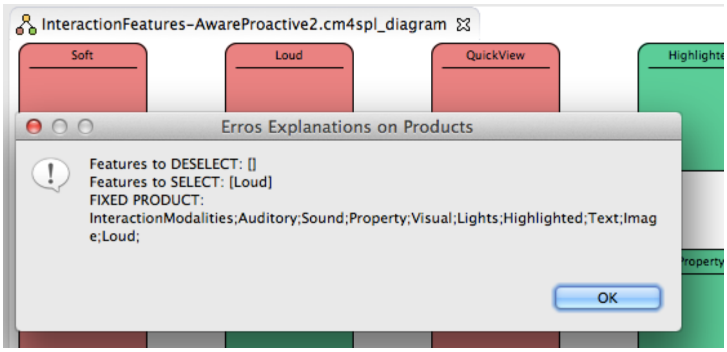
Figure A.21: FAMA operations for the Feature Model

Figure A.22 shows an example of the message after applying the model validation operation for a feature model, and Figure A.23 shows an example of the validation message for the explain errors on model operation. These operations have been integrated as plugins in the AdaptIO modeling suite, allowing designers to validate the feature model and the configurations while their specification.



The Model Is VALID

Figure A.22: Output of the model validation operation


InteractionFeatures-AwareProactive2.cm4spl_diagram
Soft
Loud
QuickView
Highlighte
Property
Erros Explanations on Products
OK

Features to DESELECT: []
Features to SELECT: [Loud]
FIXED PRODUCT:
InteractionModalities;Auditory;Sound;Property;Visual;Lights;Highlighted;Text;Imag
e;Loud;

Figure A.23: Validation message of FAMA

B

Adaptive Notifications case study & Evaluation Instruments

This appendix presents the design of the *Adaptive Notifications* case study and shows the instruments used in the different evaluations.

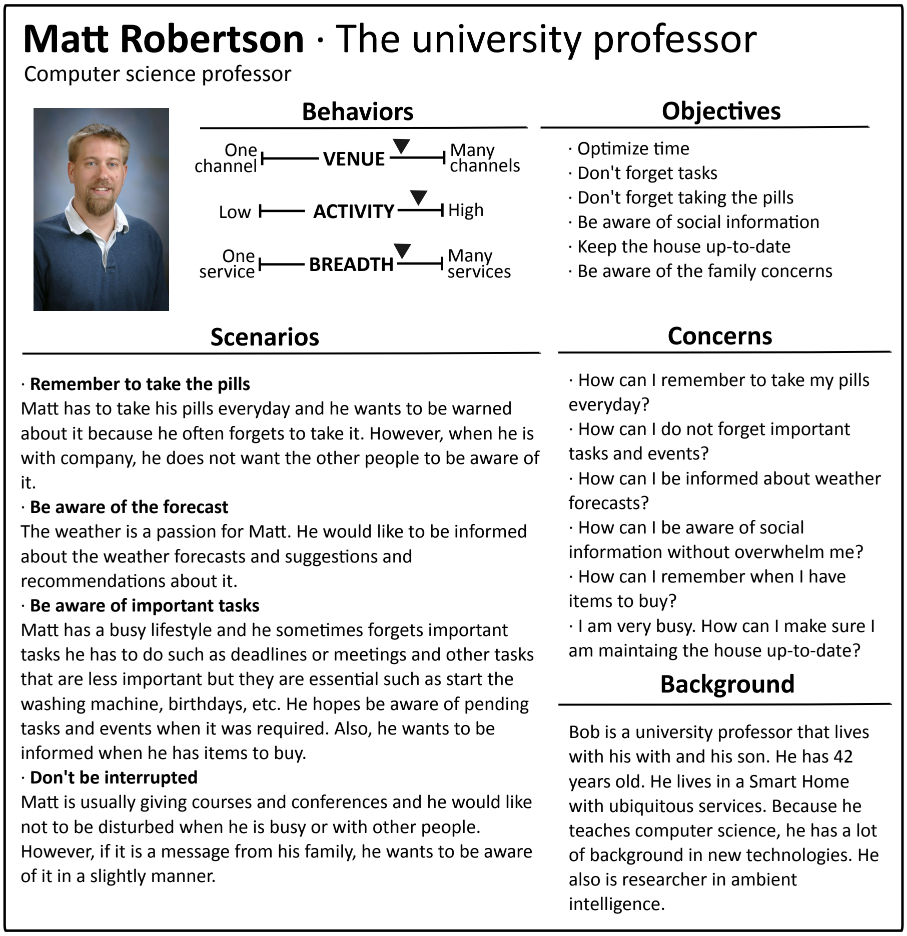
B.1 Adaptive Notifications case study

Overall, the *Adaptive Notifications* case study supports different ubiquitous services in the context of a university professor.

B.1.1 Applying our design method

In the following subsections, we describe the different aspects to be captured to specify the interaction obtrusiveness adaptation of the case study.

Persona definition



Matt Robertson · The university professor

Computer science professor

Behaviors

One channel ⊢ **VENUE** ⊣ Many channels

Low ⊢ **ACTIVITY** ⊣ High

One service ⊢ **BREADTH** ⊣ Many services

Objectives

· Optimize time
· Don't forget tasks
· Don't forget taking the pills
· Be aware of social information
· Keep the house up-to-date
· Be aware of the family concerns

Scenarios

· **Remember to take the pills**
Matt has to take his pills everyday and he wants to be warned about it because he often forgets to take it. However, when he is with company, he does not want the other people to be aware of it.

· **Be aware of the forecast**
The weather is a passion for Matt. He would like to be informed about the weather forecasts and suggestions and recommendations about it.

· **Be aware of important tasks**
Matt has a busy lifestyle and he sometimes forgets important tasks he has to do such as deadlines or meetings and other tasks that are less important but they are essential such as start the washing machine, birthdays, etc. He hopes be aware of pending tasks and events when it was required. Also, he wants to be informed when he has items to buy.

· **Don't be interrupted**
Matt is usually giving courses and conferences and he would like not to be disturbed when he is busy or with other people. However, if it is a message from his family, he wants to be aware of it in a slightly manner.

Concerns

· How can I remember to take my pills everyday?
· How can I do not forget important tasks and events?
· How can I be informed about weather forecasts?
· How can I be aware of social information without overwhelm me?
· How can I remember when I have items to buy?
· I am very busy. How can I make sure I am maintaing the house up-to-date?

Background

Bob is a university professor that lives with his with and his son. He has 42 years old. He lives in a Smart Home with ubiquitous services. Because he teaches computer science, he has a lot of background in new technologies. He also is researcher in ambient intelligence.

Figure B.1: A detailed persona

In order to give a clear picture of how users are likely to use the system and what they will expect from it we define personas. Personas capture relevant information about customers that directly impact the design process: user goals, scenarios, tasks, functionalities, and the like. Figure B.1 shows the description of the persona for Matt.

Services and obtrusiveness definition

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

Healthcare service. This service allows to manage the user health by alerting the user when he has to take his pills, and remembering to him his appointments to the doctor.

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

Home Messages service. This is a text messaging service that allows the members of the family communicate messages to the other members. This service is used in the context of the family core.

Washing Machine service. This service allows users to be informed when the washing machine is full and ready to start. Also, it notifies users when the laundry cycle is finished.

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

Weather service. This service provides information about the weather forecasts, warnings and meteorological recommendations for users.

Facebook service. This is a facebook notification service that shows users the alerts from Facebook.

Table B.1 shows the services of the system and their attentional demand needed according to the user situation (user context). Also, it shows the priority that the different services have for the user.

Service	**Attentional Demand**	**Context to Consider**
Healthcare (high priority)	medium attention, high attention	user alone, with company, working, in a meeting, in free time
Agenda (high priority)	low attention, medium attention, high attention	working, teaching, in a meeting, free-time, deadline
Home Messages (high priority)	medium attention high attention	teaching, in a meeting
Washing Machine (medium priority)	low attention, medium attention high attention	user in home, outside, sleeping, eating, cooking, watching TV
Shopping (medium priority)	medium attention, high attention	number of items to buy, user alone, with company, driving, walking, nearby supermarket
Weather (low priority)	low attention, medium attention high attention	user in home, outside, going out, with company in work, user alone
Facebook (low priority)	low attention, medium attention	user alone, with company, in free time

Table B.1: Services' analysis for the Matt Persona

Once we have defined the personas, the important services for the persona and the relevant context to consider (to form the user situations) in which the adaptation can depend on, we define the way in which services are presented in terms of obtrusiveness for the case study. This information is detailed below.

Obtrusiveness modeling

For this case study, we have defined three unobtrusive adaptation spaces in order to model the different priorities of the services. In this way, each unobtrusive adaptation space represents a specific priority. Figure B.2 shows the services of the *Adaptive Notifications* case study and their obtrusiveness level for Matt according to the different priorities of services.

The unobtrusive adaptation spaces was defined by dividing each axis in different parts as it was illustrated in Chapter 5. The attention axis is divided in three levels depending whether the interaction should be *invisible* to the user, *slightly noticeable*, or *completely aware* for the user. The initiative axis is divided in two parts that represent interactions initiated by the user (*reactive*) and interactions initiated by the system (*proactive*).

The obtrusiveness level for the different services are detailed below.

Healthcare service. The healthcare service informs the user about taking the pills when it is scheduled (*proactive* levels). The notification of this service is shown as a hint (*slightly noticeable*) when the user is with company or in a meeting. Otherwise, it is performed at the *completely aware* level of attention.

Agenda service. The notification of the agenda service is performed in a *proactive* manner in terms of initiative. Regarding the attentional demand, it moves across the attentional levels as the deadline of a notification approaches if the user is not still aware of it or he/she can be interrupted (depending on the activity he/she is engaged in) and if the priority of the notification is high. For example, if the user is attending a meeting or teaching, the notification will be provided in an *invisible* manner. However, if the user is working, suggestion will be provided in a subtle manner (*slightly* attention level). Otherwise, the notification will be provided in the *completely aware* level of attention.

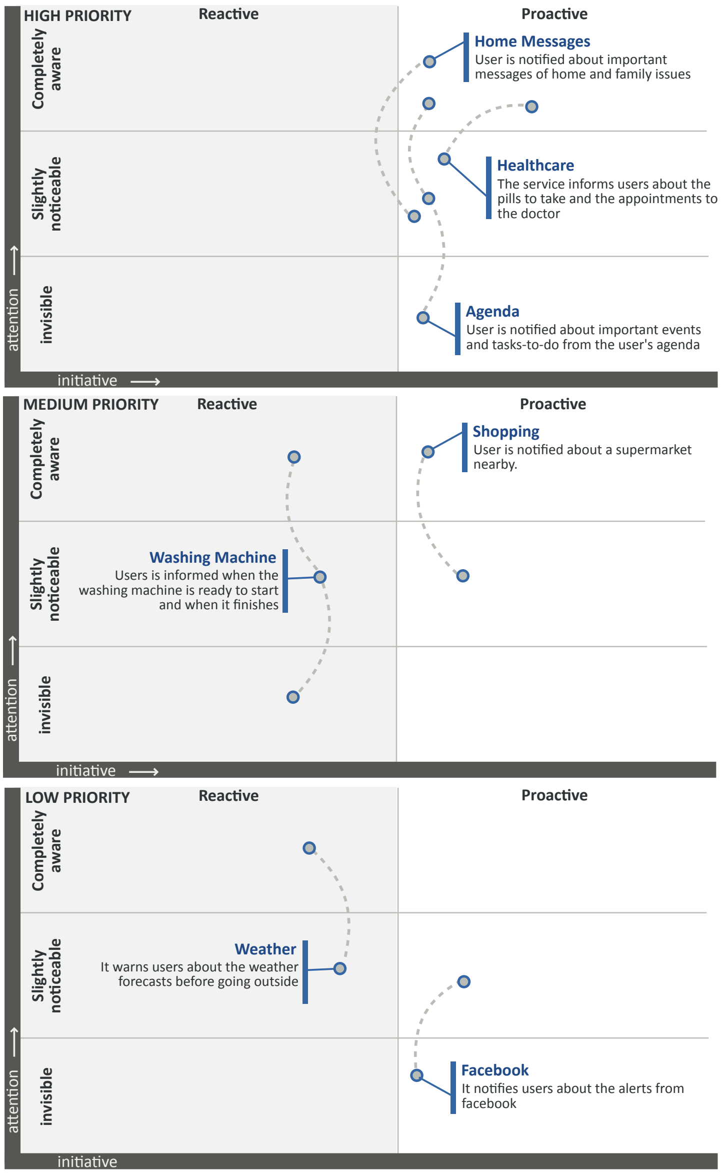

HIGH PRIORITY
Reactive
Proactive
Completely aware
Slightly noticeable
invisible
attention
initiative
Home Messages
User is notified about important messages of home and family issues
Healthcare
The service informs users about the pills to take and the appointments to the doctor
Agenda
User is notified about important events and tasks-to-do from the user's agenda
MEDIUM PRIORITY
Shopping
User is notified about a supermarket nearby.
Washing Machine
Users is informed when the washing machine is ready to start and when it finishes
LOW PRIORITY
Weather
It warns users about the weather forecasts before going outside
Facebook
It notifies users about the alerts from facebook

Figure B.2: Unobtrusive adaptation spaced defined for each service in the Adaptive Notifications case study.

Home Messages service. When the user is working, teaching or in a meeting, the messaging service passes to the *slightly appreciable* level of attention, thus reducing notification obtrusiveness when the user is working. When the system determines that the user is no longer working (in free-time), the service goes back to the *completely aware* level, increasing notification obtrusiveness.

Washing Machine service. The washing machine service notifies the user that it is ready to start after the user loads it (*reactive* levels). Notifications of this service are presented in the *invisible* level of attention when the system determines that the user is sleeping or outside home. If the user is in home, but s/he is eating or cooking, the service passes to the *slightly* level of attention. Otherwise, if the user is at home the service is presented in the *complete aware* level of attention.

Shopping service. When the user is in the proximity of a supermarket (user location), he/she is informed about a supermarket nearby. The shopping service can present notifications *proactively* in the highest level of attention (*completely aware* level) if the user is driving or walking alone and there is a supermarket nearby (e.g., by using speech feedback). However, if the user is driving with company, the notification is presented more *slightly* due to the privacy of the message.

Weather service. This service informs the user about the weather when the user leaves a building (*reactive* levels). Depending if the user is alone (*complete aware* level) or with company (*slightly* level) the service is performed in a different obtrusiveness level.

Facebook service. This service moves between the *invisible* and the *slightly* level of attention *proactively* depending if the user is in free time (slightly level) or with company (invisible level).

In order to support the behavior described above for the services, different interaction techniques can be applied. The mechanisms used

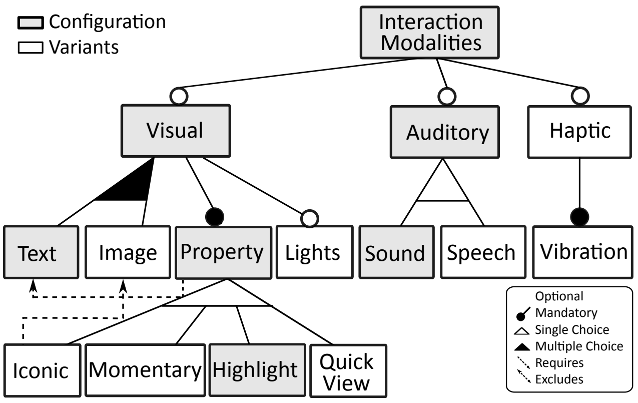


Figure B.3: Feature model of output interaction modalities.

from all the ones available for interacting with the system in this case study are described below.

Interaction variability modeling

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

Figure B.3 shows the decomposition of the available interaction in the Feature Model and the constrains for their selection. We have divided the interaction into groups of visual, auditory, and haptic modalities. These three main features include a set of manifestations of input and output modalities.

Then, we have to choose for each service the interaction features that are going to support the obtrusiveness level defined. Table B.2 shows a view of the interaction analysis results performed for the services. The table shows for each unobtrusive space, the relevant obtrusiveness level at which services can be performed and the interaction features selected

Unobtrusive Space	**Obtrusiveness Level**	**Interaction Features**
High priority	(proactive, invisible)	iconic, image
High priority	(proactive, slightly)	quick view, sound
High priority	(proactive, aware)	highlight, speech
Medium priority	(reactive, invisible)	momentary
Medium priority	(reactive, slightly)	iconic, vibration
Medium priority	(reactive, aware)	quick view, sound
Medium priority	(proactive, slightly)	iconic, vibration
Medium priority	(proactive, aware)	quick view, speech
Low priority	(reactive, slightly)	momentary, vibration
Low priority	(reactive, aware)	iconic, sound
Low priority	(proactive, invisible)	-
Low priority	(proactive, slightly)	iconic, image

Table B.2: Interaction features for each obtrusiveness level of the unobtrusive adaptation spaces

for the obtrusiveness level.

Concrete interaction modeling

In order to define the concrete user interface components that support the interaction techniques available we define the node tree of our system. For the *Adaptive Notifications* case study, the concrete components are shown in Figure B.4.

The concrete UI components that support the different interaction features are specified in Table B.3

B.2 Instruments used in the evaluations

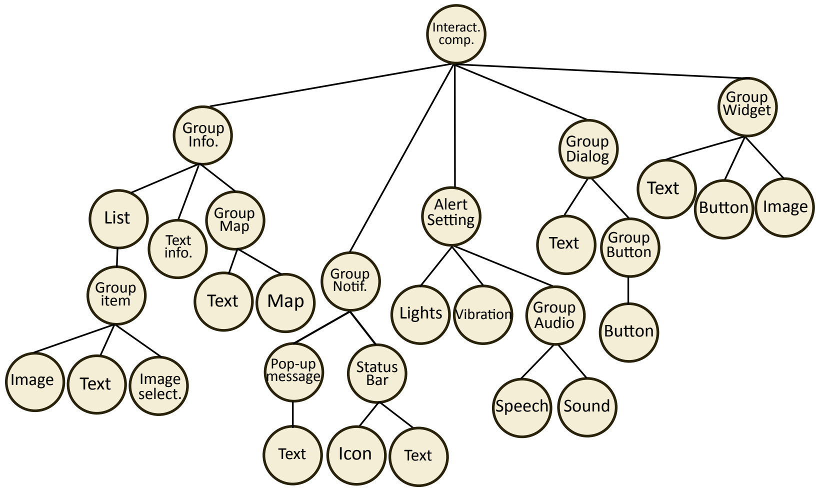


Figure B.4: Concrete Interaction Components model.

Interaction feature	**Concrete components**
Iconic	Group Notif., Status Bar
Momentary	Group Notif. Pop-up message
Highlight	Group Dialog
Quick View	Group Widget
Lights	Alert Setting, Lights
Sound	Group Audio, Sound
Speech	Group Audio, Speech
Vibration	Alert Setting, Vibration

Table B.3: Linking between interaction features and concrete components

In this section, the different questionnaires used in the evaluations are presented.

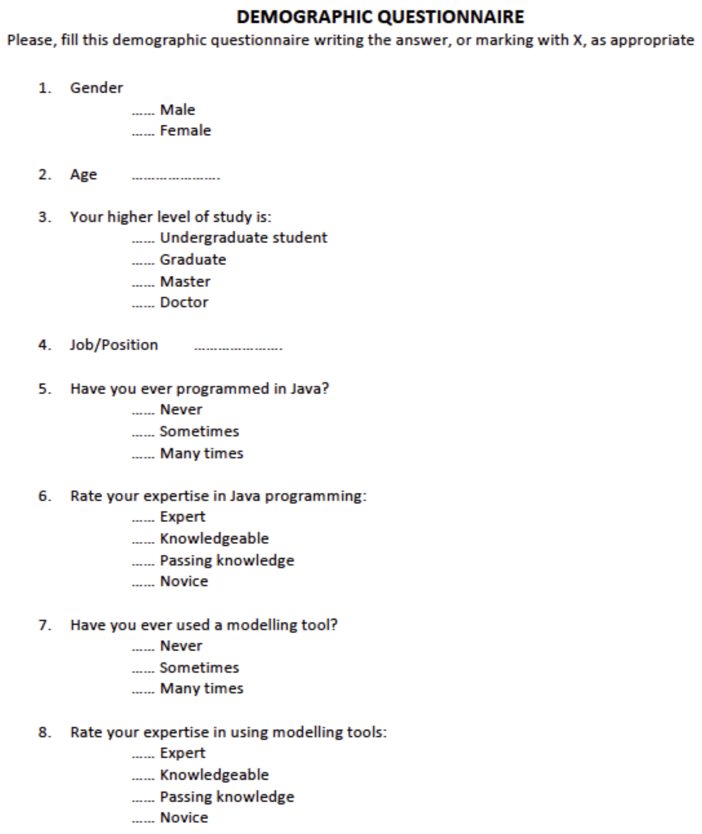
B.2.1 Usability evaluation of the design method

The questionnaires that were used to carry out this experiment were:

1. a *demographic questionnaire* to know the level of the users' experience in Java programming and modeling tools

2. a *survey* with a list of questions defined to capture the duration times of each task and the users' perceptions in a 7-Likert scale format. The perceived satisfaction was captured using the two perceptions of satisfaction (system usefulness and overall satisfaction) of the CSUQ questionnaire (Lewis, 1995).

These instruments are illustrated in the following pages.



DEMOGRAPHIC QUESTIONNAIRE

Please, fill this demographic questionnaire writing the answer, or marking with X, as appropriate

1. Gender
   ...... Male
   ...... Female

2. Age ....

3. Your higher level of study is:
   ...... Undergraduate student
   ...... Graduate
   ...... Master
   ...... Doctor

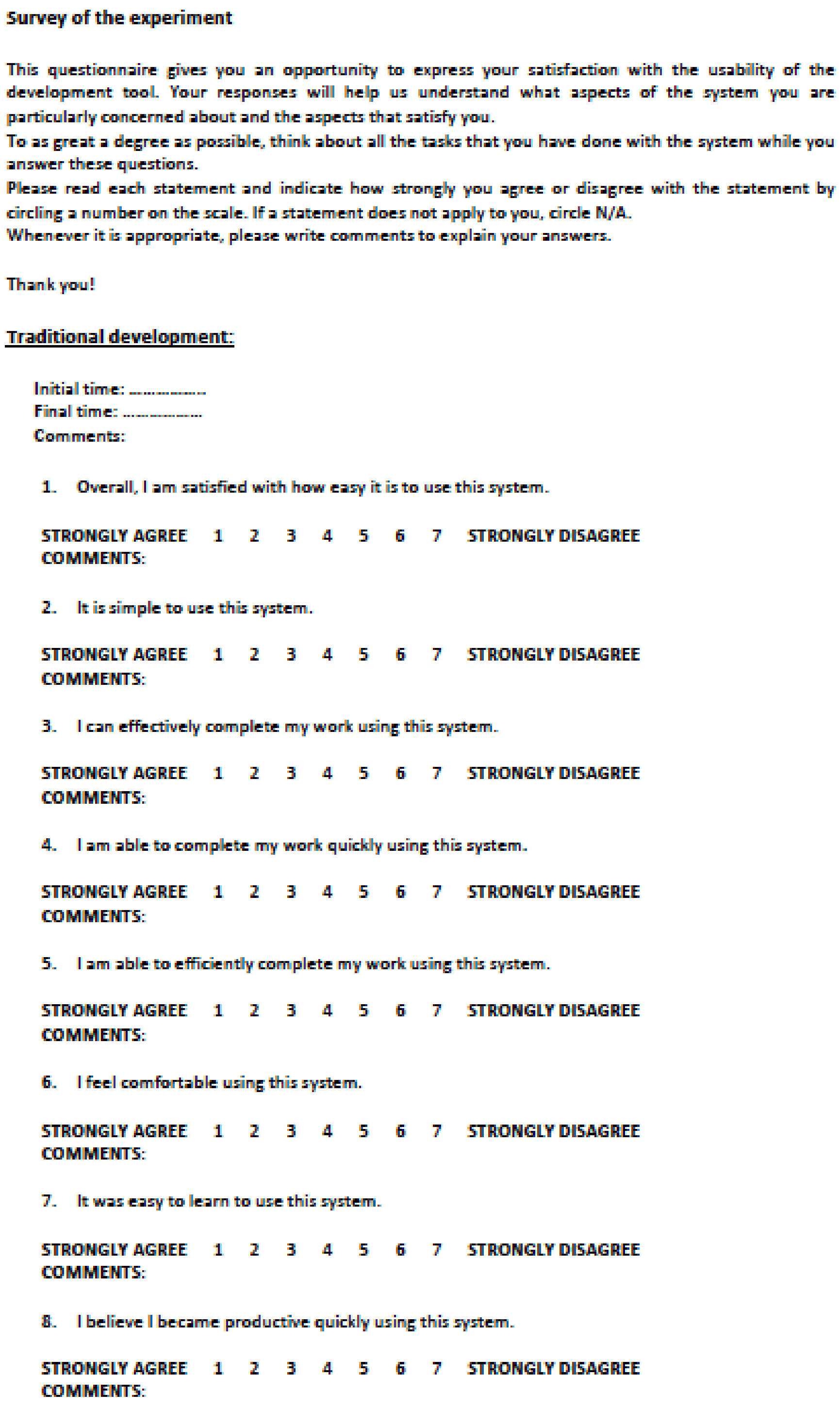
4. Job/Position ....

5. Have you ever programmed in Java?
   ...... Never
   ...... Sometimes
   ...... Many times

6. Rate your expertise in Java programming:
   ...... Expert
   ...... Knowledgeable
   ...... Passing knowledge
   ...... Novice

7. Have you ever used a modelling tool?
   ...... Never
   ...... Sometimes
   ...... Many times

8. Rate your expertise in using modelling tools:
   ...... Expert
   ...... Knowledgeable
   ...... Passing knowledge
   ...... Novice



Survey of the experiment

This questionnaire gives you an opportunity to express your satisfaction with the usability of the development tool. Your responses will help us understand what aspects of the system you are particularly concerned about and the aspects that satisfy you.
To as great a degree as possible, think about all the tasks that you have done with the system while you answer these questions.
Please read each statement and indicate how strongly you agree or disagree with the statement by circling a number on the scale. If a statement does not apply to you, circle N/A.
Whenever it is appropriate, please write comments to explain your answers.

Thank you!

Traditional development:

Initial time: ……………
Final time: ……………
Comments:

1. Overall, I am satisfied with how easy it is to use this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

2. It is simple to use this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

3. I can effectively complete my work using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

4. I am able to complete my work quickly using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

5. I am able to efficiently complete my work using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

6. I feel comfortable using this system.

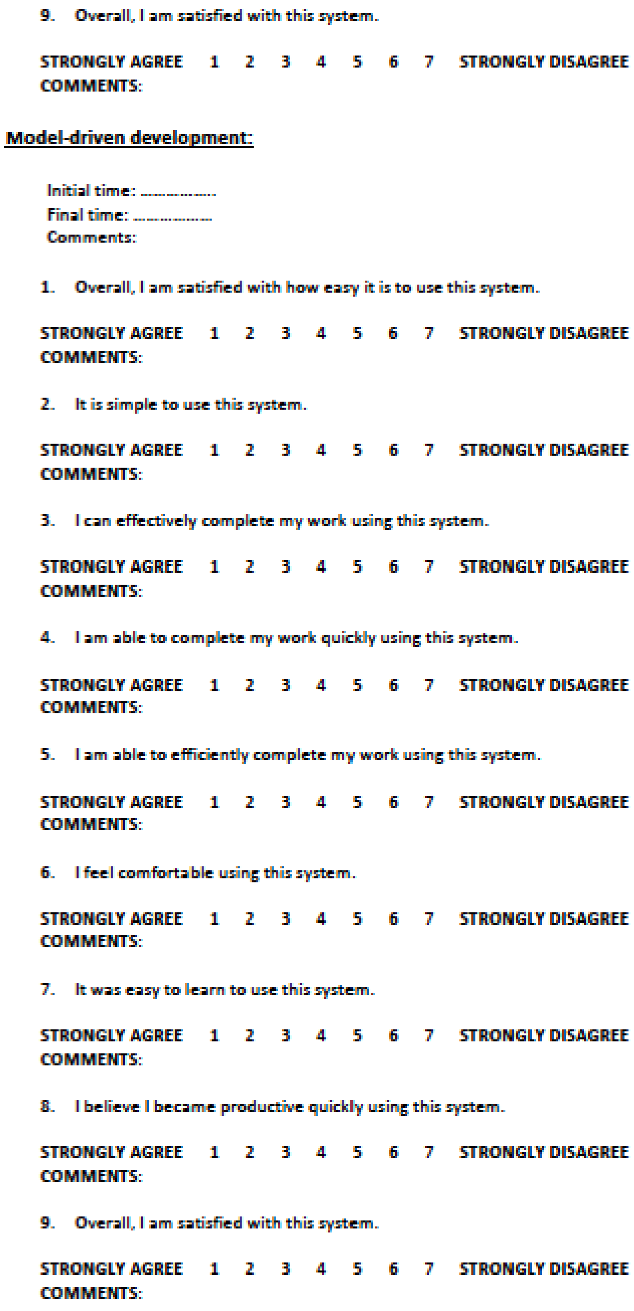
STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

7. It was easy to learn to use this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

8. I believe I became productive quickly using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:



9. Overall, I am satisfied with this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

Model-driven development:

Initial time: ....
Final time: ....
Comments:

1. Overall, I am satisfied with how easy it is to use this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

2. It is simple to use this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

3. I can effectively complete my work using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

4. I am able to complete my work quickly using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

5. I am able to efficiently complete my work using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

6. I feel comfortable using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

7. It was easy to learn to use this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

8. I believe I became productive quickly using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

9. Overall, I am satisfied with this system.

STRONGLY AGREE 1 2 3 4 5 6 7 STRONGLY DISAGREE
COMMENTS:

B.2.2 User satisfaction evaluation

In order to evaluate the user satisfaction in the simulation of the *Smart Home* case study, we used an adapted IBM Post-Study questionnaire (Lewis, 1995) in conjunction with the questionnaire defined by Vastenburg et al. in (Vastenburg et al., 2008) to evaluate home notification systems for three dimensions. This three dimensions were:

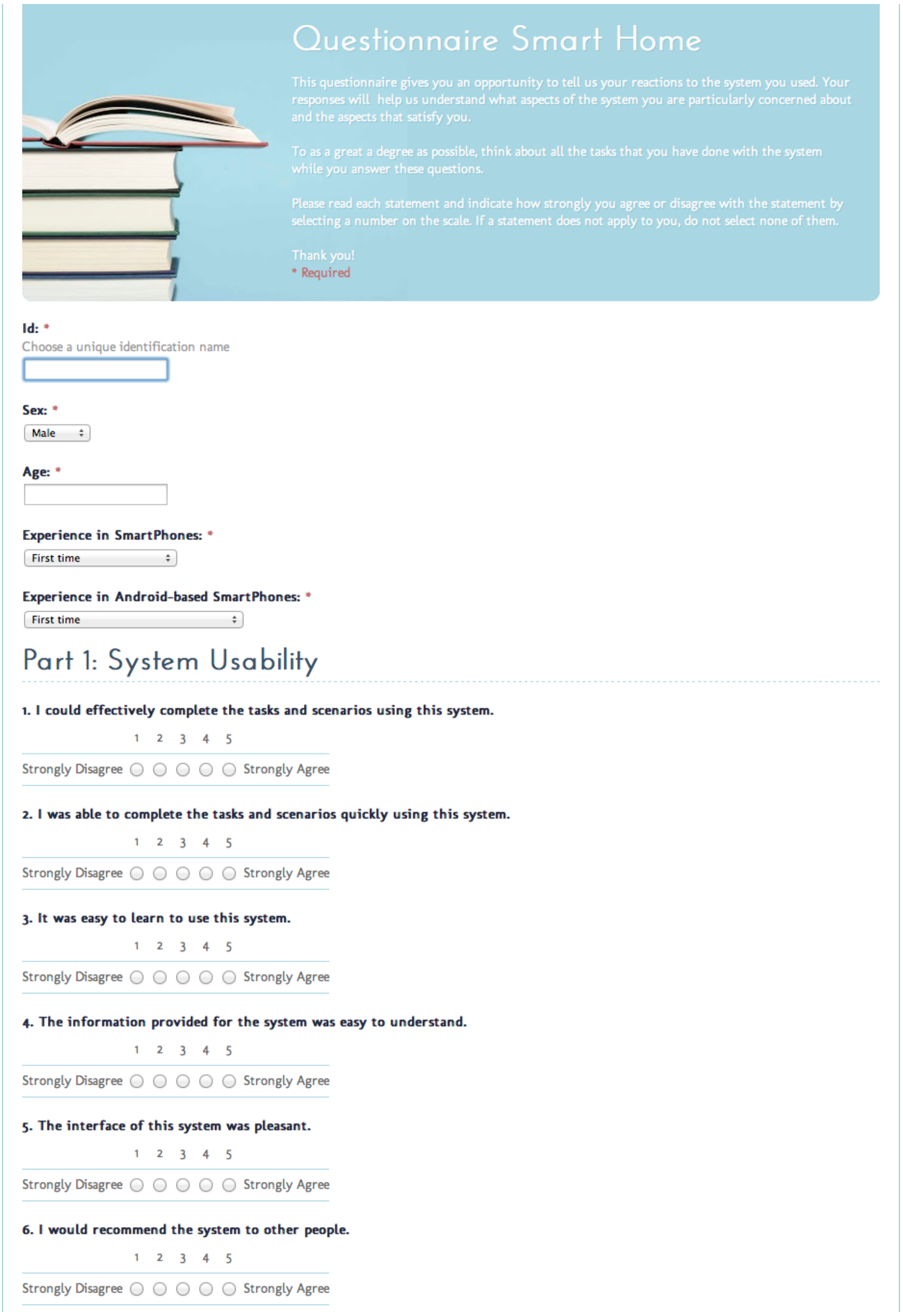
- *usability of the system*
- *messages acceptability*
- *interaction adaptation*

The first dimension focuses on measuring users' acceptance with the usability of the system; the second one focuses on the general acceptability considering the messages and the user activity at the time of notification; and finally, the third dimension is about users' satisfaction in the interaction adaptation.

Furthermore, in order to analyze how our system is capable of handling the attention resources of the user by means of the adaptations, we used the NASA Task Load Index [1] (NASA-TLX) to obtain the subjective cognitive load level (SCL). TLX contains six workload-related dimensions: mental demand, physical demand, temporal demand, own performance, effort, and frustration. A 10-level rating can be performed on each of the six dimensions.

These questionnaires are illustrated in the following pages.

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



Questionnaire Smart Home

This questionnaire gives you an opportunity to tell us your reactions to the system you used. Your responses will help us understand what aspects of the system you are particularly concerned about and the aspects that satisfy you.

To as a great a degree as possible, think about all the tasks that you have done with the system while you answer these questions.

Please read each statement and indicate how strongly you agree or disagree with the statement by selecting a number on the scale. If a statement does not apply to you, do not select none of them.

Thank you!
* Required

Id: *
Choose a unique identification name

Sex: *
Male

Age: *

Experience in SmartPhones: *
First time

Experience in Android-based SmartPhones: *
First time

Part 1: System Usability

1. I could effectively complete the tasks and scenarios using this system.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

2. I was able to complete the tasks and scenarios quickly using this system.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

3. It was easy to learn to use this system.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

4. The information provided for the system was easy to understand.

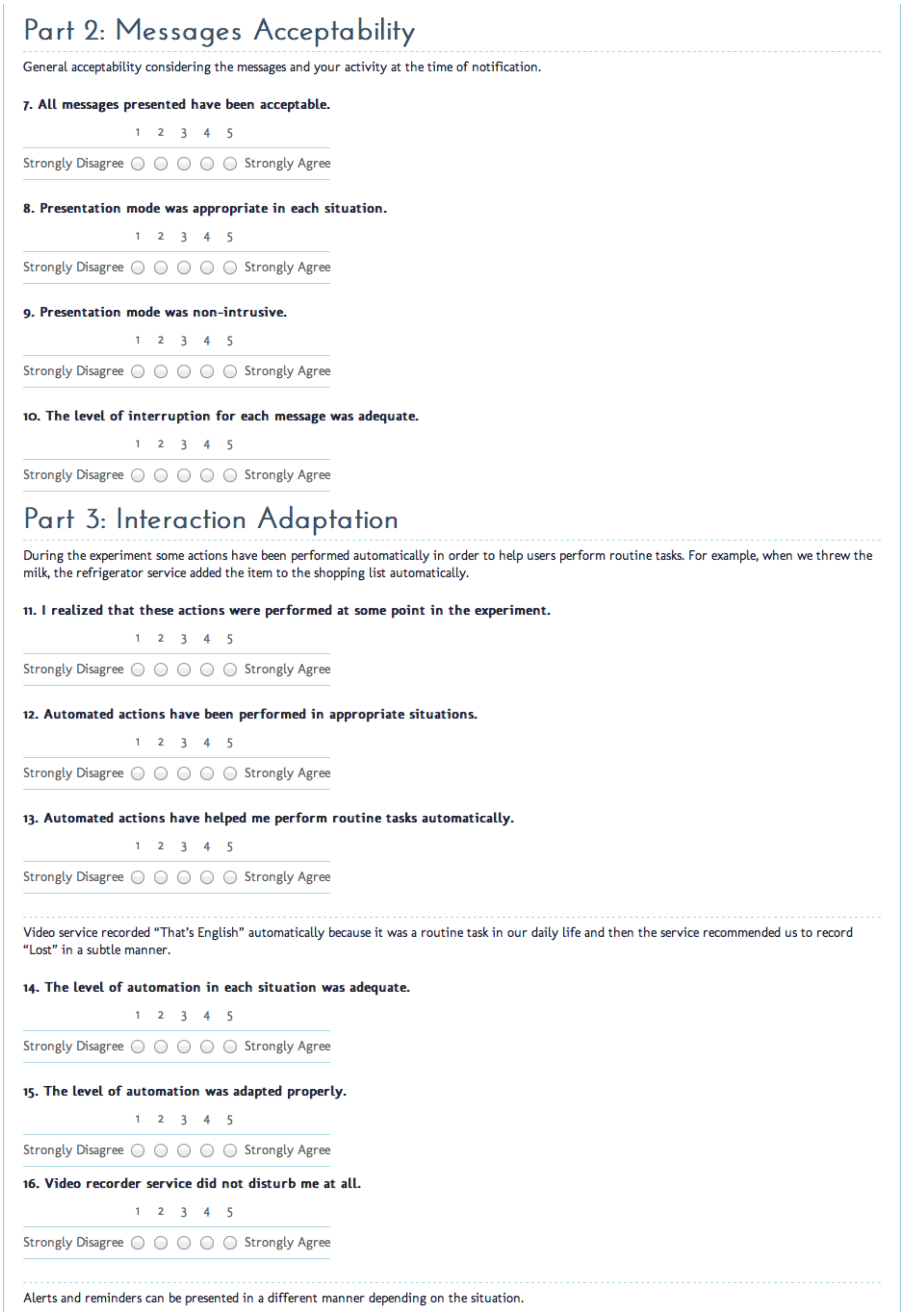
	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

5. The interface of this system was pleasant.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

6. I would recommend the system to other people.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree



Part 2: Messages Acceptability

General acceptability considering the messages and your activity at the time of notification.

7. All messages presented have been acceptable.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

8. Presentation mode was appropriate in each situation.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

9. Presentation mode was non-intrusive.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

10. The level of interruption for each message was adequate.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Part 3: Interaction Adaptation

During the experiment some actions have been performed automatically in order to help users perform routine tasks. For example, when we threw the milk, the refrigerator service added the item to the shopping list automatically.

11. I realized that these actions were performed at some point in the experiment.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

12. Automated actions have been performed in appropriate situations.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

13. Automated actions have helped me perform routine tasks automatically.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Video service recorded "That's English" automatically because it was a routine task in our daily life and then the service recommended us to record "Lost" in a subtle manner.

14. The level of automation in each situation was adequate.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

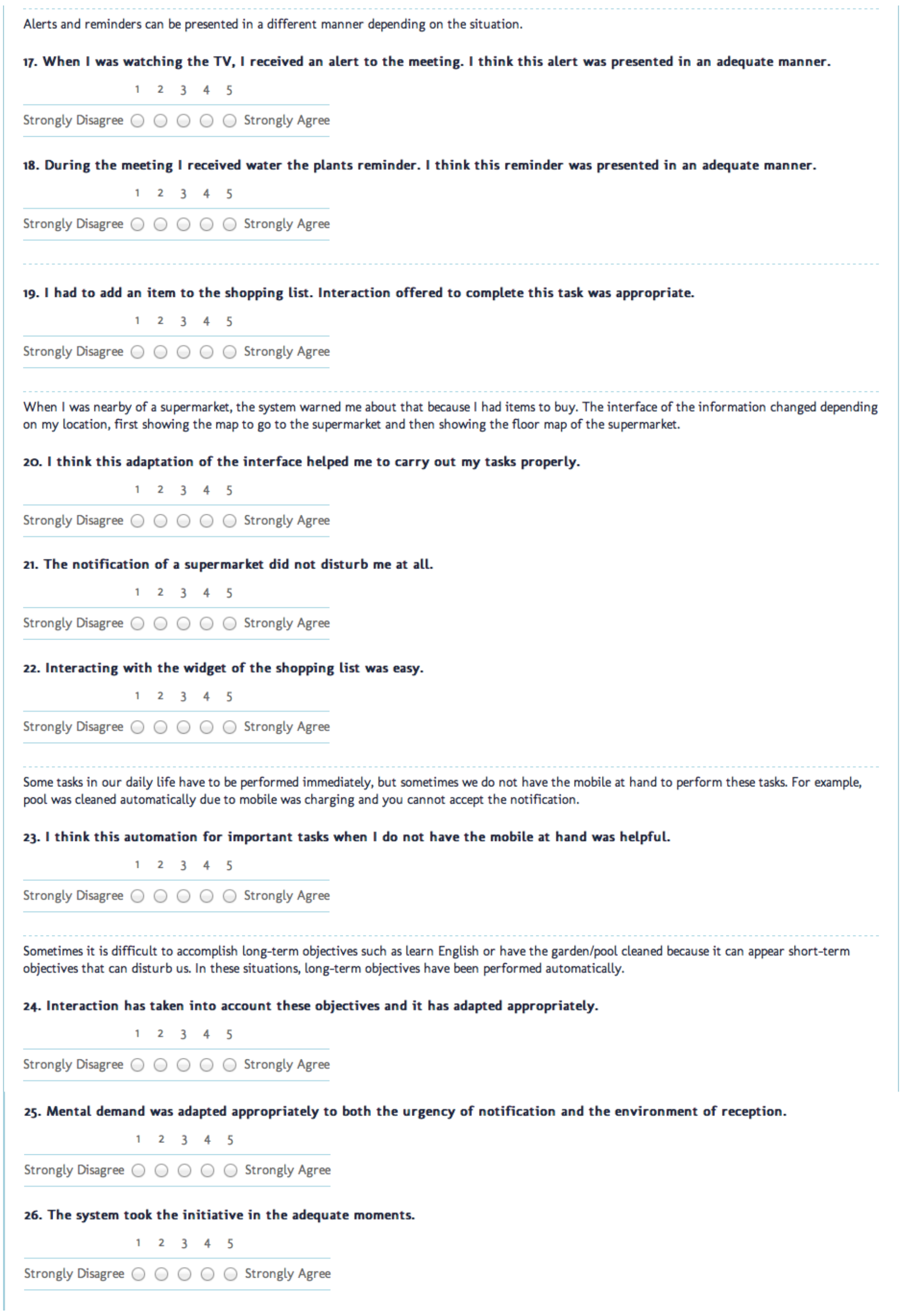
15. The level of automation was adapted properly.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

16. Video recorder service did not disturb me at all.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Alerts and reminders can be presented in a different manner depending on the situation.



Alerts and reminders can be presented in a different manner depending on the situation.

17. When I was watching the TV, I received an alert to the meeting. I think this alert was presented in an adequate manner.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

18. During the meeting I received water the plants reminder. I think this reminder was presented in an adequate manner.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

19. I had to add an item to the shopping list. Interaction offered to complete this task was appropriate.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

When I was nearby of a supermarket, the system warned me about that because I had items to buy. The interface of the information changed depending on my location, first showing the map to go to the supermarket and then showing the floor map of the supermarket.

20. I think this adaptation of the interface helped me to carry out my tasks properly.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

21. The notification of a supermarket did not disturb me at all.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

22. Interacting with the widget of the shopping list was easy.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Some tasks in our daily life have to be performed immediately, but sometimes we do not have the mobile at hand to perform these tasks. For example, pool was cleaned automatically due to mobile was charging and you cannot accept the notification.

23. I think this automation for important tasks when I do not have the mobile at hand was helpful.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Sometimes it is difficult to accomplish long-term objectives such as learn English or have the garden/pool cleaned because it can appear short-term objectives that can disturb us. In these situations, long-term objectives have been performed automatically.

24. Interaction has taken into account these objectives and it has adapted appropriately.

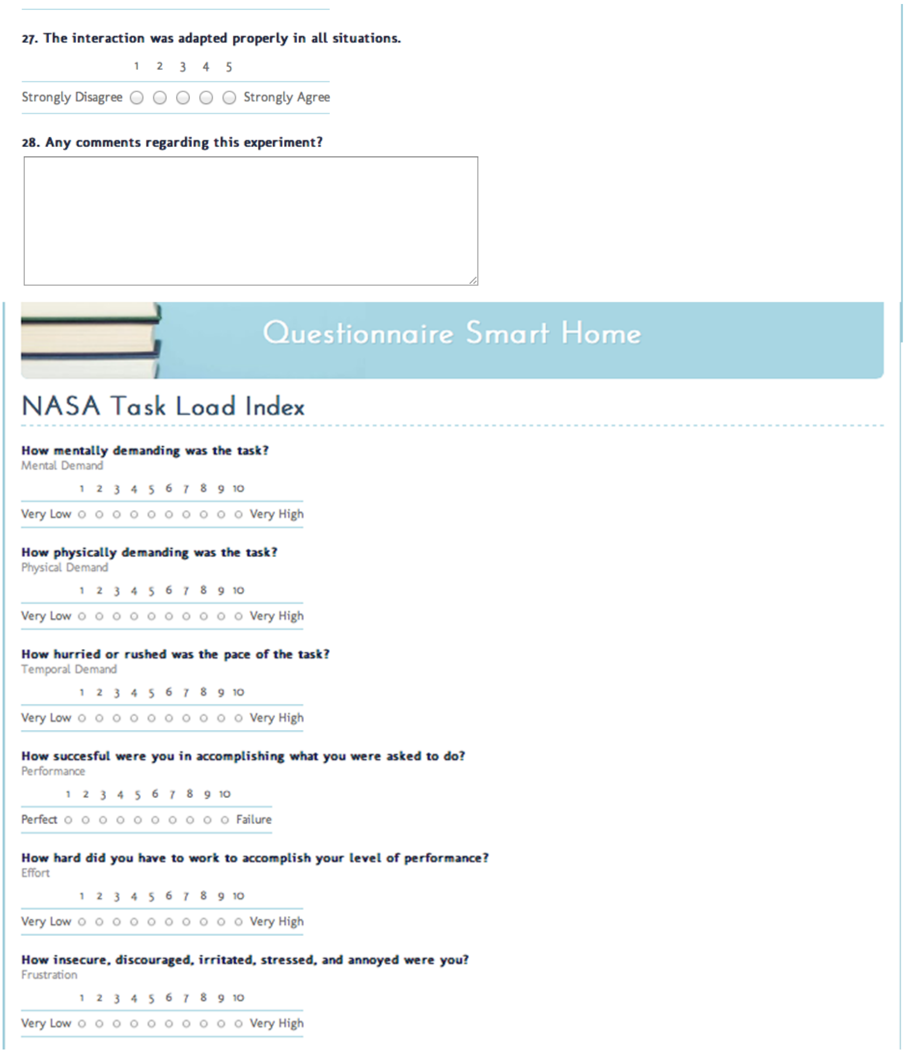
	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

25. Mental demand was adapted appropriately to both the urgency of notification and the environment of reception.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

26. The system took the initiative in the adequate moments.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree



27. The interaction was adapted properly in all situations.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

28. Any comments regarding this experiment?

Questionnaire Smart Home

NASA Task Load Index

How mentally demanding was the task?
Mental Demand

	1	2	3	4	5	6	7	8	9	10	
Very Low	○	○	○	○	○	○	○	○	○	○	Very High

How physically demanding was the task?
Physical Demand

	1	2	3	4	5	6	7	8	9	10	
Very Low	○	○	○	○	○	○	○	○	○	○	Very High

How hurried or rushed was the pace of the task?
Temporal Demand

	1	2	3	4	5	6	7	8	9	10	
Very Low	○	○	○	○	○	○	○	○	○	○	Very High

How succesful were you in accomplishing what you were asked to do?
Performance

	1	2	3	4	5	6	7	8	9	10	
Perfect	○	○	○	○	○	○	○	○	○	○	Failure

How hard did you have to work to accomplish your level of performance?
Effort

	1	2	3	4	5	6	7	8	9	10	
Very Low	○	○	○	○	○	○	○	○	○	○	Very High

How insecure, discouraged, irritated, stressed, and annoyed were you?
Frustration

	1	2	3	4	5	6	7	8	9	10	
Very Low	○	○	○	○	○	○	○	○	○	○	Very High

B.2.3 User experience evaluation

In order to measure the User Experience, we used the AttrakDiff2 questionnaire. This questionnaire is an instrument for measuring the attractiveness of interactive products. With the help of pairs of opposite adjectives, users (or potential users) can indicate their perception of the product. these adjective-pairs make a collation of the evaluation dimensions possible. The following product dimensions are evaluated:

- *Pragmatic Quality (PQ)*: Describes the usability of a product and indicates how successfully users are in achieving their goals using the product.

- *Hedonic Quality - Stimulation (HQ-S)*: Mankind has an inherent need to develop and move forward. This dimension indicates to what extent the product can support those needs in terms of novel, interesting, and stimulating functions, contents, and interaction- and presentation-styles.

- *Hedonic Quality - Identity (HQ-I)*: Indicates to what extent the product allows the user to identify with it.

- *Attractiveness (ATT)*: Describes a global value of the product based on the quality perception.

Hedonic and pragmatic qualities are independent of one another, and contribute equally to the rating of attractiveness. This questionnaire is illustrated below.

Assessment of the system

Please state your impression of the system you tested by means of the following pairs of words:

technical	☐	☐	☐	☐	☐	☐	☐	human
complicated	☐	☐	☐	☐	☐	☐	☐	simple
impractical	☐	☐	☐	☐	☐	☐	☐	practical
cumbersome	☐	☐	☐	☐	☐	☐	☐	straightforward
unpredictable	☐	☐	☐	☐	☐	☐	☐	predictable
confusing	☐	☐	☐	☐	☐	☐	☐	clearly structured
unruly	☐	☐	☐	☐	☐	☐	☐	manageable
isolating	☐	☐	☐	☐	☐	☐	☐	connective
unprofessional	☐	☐	☐	☐	☐	☐	☐	professional
tacky	☐	☐	☐	☐	☐	☐	☐	stylish
cheap	☐	☐	☐	☐	☐	☐	☐	premium
alienating	☐	☐	☐	☐	☐	☐	☐	integrating
separates me	☐	☐	☐	☐	☐	☐	☐	brings me closer
unpresentable	☐	☐	☐	☐	☐	☐	☐	presentable
conventional	☐	☐	☐	☐	☐	☐	☐	inventive
unimaginative	☐	☐	☐	☐	☐	☐	☐	creative
cautious	☐	☐	☐	☐	☐	☐	☐	bold
conservative	☐	☐	☐	☐	☐	☐	☐	innovative
dull	☐	☐	☐	☐	☐	☐	☐	captivating
undemanding	☐	☐	☐	☐	☐	☐	☐	challenging
ordinary	☐	☐	☐	☐	☐	☐	☐	novel
unpleasant	☐	☐	☐	☐	☐	☐	☐	pleasant
ugly	☐	☐	☐	☐	☐	☐	☐	attractive
disagreeable	☐	☐	☐	☐	☐	☐	☐	likeable
rejecting	☐	☐	☐	☐	☐	☐	☐	inviting
bad	☐	☐	☐	☐	☐	☐	☐	good
repelling	☐	☐	☐	☐	☐	☐	☐	apelling
discouraging	☐	☐	☐	☐	☐	☐	☐	motivating

B.2.4 Customization interfaces evaluation

System Usability questionnaire

This questionnaire is a 19-item instrument for assessing user satisfaction with system usability. It gives you an opportunity to tell us your reactions to the system you used. Your responses will help us understand what aspects of the system you are particularly concerned about and the aspects that satisfy you.

To as a great a degree as possible, think about all the tasks that you have done with the system while you answer these questions.

Please read each statement and indicate how strongly you agree or disagree with the statement by selecting a number on the scale.

Thank you!
* Required

Id:* …………

Sex:*

☐ Male
☐ Female

Age:*………..

Computer knowledge: ……………..

Experience in Smartphones:*

☐ First time
☐ I have used it sometime
☐ I own a smartphone less than 1 year
☐ I own a smartphone more than 1 year

Experience in Android-based Smartphones:*

☐ First time
☐ I have used it sometime
☐ I own an Android-based smartphone less than 1 year
☐ I own an Android-based smartphone more than 1 year

1. Overall, I am satisfied with how easy it is to use this system.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

2. It is simple to use this system.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

3. I could effectively complete the tasks and scenarios using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

4. I was able to complete the tasks and scenarios quickly using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

5. I was able to efficiently complete the tasks and scenarios using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

6. I felt comfortable using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

7. It was easy to learn to use this system.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

8. I believe I could become productive quickly using this system.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

9. The system gave me error messages that clearly told me how to fix problems.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

10. Whenever I made a mistake using the system, I could recover easily and quickly.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

11. The information provided with this system was clear.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

12. It was easy to find the information I needed.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

13. The information provided for the system was easy to understand.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

14. The information was effective in helping me complete the tasks and scenarios.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

15. The organization of information on the system screens was clear.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

16. The interface of this system was pleasant.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

17. I liked using the interface of this system.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

18. This system has all the functions and capabilities I expect it to have.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

19. Overall, I am satisfied with this system.

STRONGLY AGREE 1 2 3 4 5 6 7 **STRONGLY DISAGREE**

ADDITIONAL COMMENTS:

Bibliography

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

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

Bachvarova, Y., van Dijk, B., and Nijholt, A. (2007). Towards a unified knowledge-based approach to modality choice. In *Proc. Workshop on Multimodal Output Generation (MOG)*, pages 5–15.

Balme, L., Demeure, A., Barralon, N., Coutaz, J., and Calvary, G. (2004). Cameleon-rt: A software architecture reference model for distributed, migratable, and plastic user interfaces. In *Ambient Intelligence*, volume 3295 of *Lecture Notes in Computer Science*, pages 291–302.

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

Benavides, D., Trinidad, P., and Ruiz-Cortés, A. (2005). Automated reasoning on feature models. In *Proceedings of the 17th international conference on Advanced Information Systems Engineering*, CAiSE'05, pages 491–503, Berlin, Heidelberg. Springer-Verlag.

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

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., and Riboni, D. (2010). A survey of context modelling and reasoning techniques. *Pervasive Mob. Comput.*, 6(2):161–180.

Bezivin, J. and Gerbe, O. (2001). Towards a precise definition of the omg/mda framework. In *Automated Software Engineering, 2001. (ASE 2001). Proceedings. 16th Annual International Conference on*, pages 273 – 280.

Blumendorf, M., Lehmann, G., and Albayrak, S. (2010a). 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, pages 9–18, New York, NY, USA. ACM.

Blumendorf, M., Lehmann, G., Roscher, D., and Albayrak, S. (2010b). *Ubiquitous User Interfaces: Multimodal Adaptive Interaction for Smart Environments*, pages 24–52. IGI Global.

Broll, G., Haarländer, M., Paolucci, M., Wagner, M., Rukzio, E., and Schmidt, A. (2008). Collect&drop: A technique for multi-tag interaction with real world objects and information. In *AmI*, pages 175–191.

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

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

Butter, T., Aleksy, M., Bostan, P., and Schader, M. (2007). Context-aware user interface framework for mobile applications. In *Distributed Computing Systems Workshops, 2007. ICDCSW '07. 27th International Conference on*, page 39.

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

Buxton, W. (1990). Human-computer interaction. chapter There's more to interaction than meets the eye: some issues in manual input, pages 122–137. Prentice Hall Press, Upper Saddle River, NJ, USA.

Byun, H. E. and Cheverst, K. (2001a). Exploiting user models and context-awareness to support personal daily activities. In *Personal Daily Activities, Workshop in UM2001 on User Modelling for Context-Aware Applications*, pages 13–16.

Byun, H. E. and Cheverst, K. (2001b). Exploiting user models and context-awareness to support personal daily activities. In *Workshop in UM2001 on User Modelling for Context-Aware Applications, Sonthofen, Germany.*

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

Cao, Y., Theune, M., and Nijholt, A. (2009). 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, pages 335–344, New York, NY, USA. ACM.

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

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

Chen, H. (2004). *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland, Baltimore County.

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

Chin, J., Callaghan, V., and Clarke, G. (2006). An end-user programming paradigm for pervasive computing applications. In *Pervasive Services, 2006 ACS/IEEE International Conference on*, pages 325–328.

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

Clark, H. (1996). *Using Language*. Cambridge University Press.

Clerckx, T., Vandervelpen, C., and Coninx, K. (2008). Task-based design and runtime support for multimodal user interface distribution. In *Engineering Interactive Systems*, volume 4940 of *Lecture Notes in Computer Science*, pages 89–105.

Cohen, M. H., Giangola, J. P., and Balogh, J. (2004). *Voice User Interface Design*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Coninx, K., Luyten, K., Vandervelpen, C., den Bergh, J. V., and Creemers, B. (2003). Dygimes: Dynamically generating interfaces for mobile computing devices and embedded systems. In *Human-Computer Interaction with Mobile Devices and Services*, volume 2795 of *Lecture Notes in Computer Science*, pages 256–270.

Constantine, L. L. (2009). Interaction design and model-driven development. In *Proceedings of the 12th International Conference on*

Model Driven Engineering Languages and Systems, MODELS '09, pages 377–377, Berlin, Heidelberg. Springer-Verlag.

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

Czarnecki, K., Helsen, S., and Eisenecker, U. (2004). Staged configuration using feature models. In *Software Product Lines*, volume 3154 of *Lecture Notes in Computer Science*, pages 162–164.

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

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

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

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

de Sá, M., Duarte, C., Carriço, L., and Reis, T. (2010). *Desining Mobile Multimodal Applications*, chapter 5, pages 106–136. Information Science Reference.

den Bergh, J. V. and Coninx, K. (2006). Cup 2.0: High-level modeling of context-sensitive interactive applications. In Nierstrasz, O., Whittle, J., Harel, D., and Reggio, G., editors, *MoDELS*, volume 4199 of *Lecture Notes in Computer Science*, pages 140–154. Springer.

Dey, A. K. and Abowd, G. D. (2000). Towards a better understanding of context and context-awareness. In *Workshop on The What, Who,* *Where, When, and How of Context-Awareness (CHI 2000)*, The Hague, The Netherlands.

Dey, A. K. and Häkkilä, J. (2008). Context-awareness and mobile devices. chapter XIII, pages 205–217. IGI Global.

Dey, A. K., Hamid, R., Beckmann, C., Li, I., and Hsu, D. (2004). a cappella: programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 33–40, New York, NY, USA. ACM.

Dix, A., Finlay, J. E., Abowd, G. D., and Beale, R. (2003). *Human-Computer Interaction (3rd Edition)*. Prentice Hall, New York, NY, USA.

Doctor, F., Hagras, H., and Callaghan, V. (2005). An intelligent fuzzy agent approach for realising ambient intelligence in intelligent inhabited environments. *IEEE Transactions on System, Man & Cybernetics*, 35(1):55–65.

Dourish, P. (2004). What we talk about when we talk about context. *Personal Ubiquitous Comput.*, 8(1):19–30.

Dowling, J., Cunningham, R., Curran, E., and Cahill, V. (2006). Building autonomic systems using collaborative reinforcement learning. *Knowl. Eng. Rev.*, 21(3):231–238.

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

Dubberly, H., Pangaro, P., and Haque, U. (2009). On modeling: What is interaction?: are there different types? *interactions*, 16(1):69–75.

Endsley, M. and Kaber, D. (1999). Level of automation effects on performance, situation awareness and workload in a dynamic control task. *Ergonomics*, 42(3):462–492.

Fabro, M. D. D., Bézivin, J., and Valduriez, P. (2006). Weaving models with the eclipse amw plugin. *Eclipse Modeling Symposium.*

Favre, J.-M. (2004). Foundations of Model (Driven) (Reverse) Engineering : Models – Episode I: Stories of the fidus papyrus and of the solarus. In Bezivin, J. and Heckel, R., editors, *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

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

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

Fischer, J. E., Yee, N., Bellotti, V., Good, N., Benford, S., and Greenhalgh, C. (2010). Effects of content and time of delivery on receptivity to mobile interruptions. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, MobileHCI '10, pages 103–112, New York, NY, USA. ACM.

Flyvbjerg, B. (2006). Five Misunderstandings About Case-Study Research. *Qualitative Inquiry*, 12(2):219–245.

France, R. and Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 37–54, Washington, DC, USA. IEEE Computer Society.

Gershenfeld, N. (2000). *When Things Start to Think.* Owl Books, New York, NY, USA.

Gershenfeld, N., Krikorian, R., and Cohen, D. (2004). The Internet of Things. *Scientific American*, 291(4):46–51.

Gibbs, W. W. (2005). Considerate computing. *Scientific American*, 292(1):54–61.

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

Ginsburg, G. (2004). *Interruptions: A Criterion in the Design and Evaluation of Human-Computer Interfaces*. PhD thesis, University of Toronto.

Godoy, D. and Amandi, A. (2005). User profiling for web page filtering. *IEEE Internet Computing*, 9(4):56–64.

Grandhi, S. and Jones, Q. (2009). Conceptualizing interpersonal interruption management: A theoretical framework and research program. In *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, pages 1–10.

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

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220.

Grundy, J. and Hosking, J. (2000). Developing adaptable user interfaces for component-based systems. In *Proceedings of the First Australasian User Interface Conference*, AUIC '00, pages 17–, Washington, DC, USA. IEEE Computer Society.

Gulliksen, J., Göransson, B., Boivie, I., Persson, J., Blomkvist, S., and Cajander, Å. (2005). Key principles for user-centred systems design. In Seffah, A., Gulliksen, J., and Desmarais, M. C., editors, *Human-Centered Software Engineering — Integrating Usability in the Software Development Lifecycle*, volume 8 of *Human-Computer Interaction Series*, pages 17–36. Springer Netherlands.

Haapalainen, E., Kim, S., Forlizzi, J. F., and Dey, A. K. (2010). Psycho-physiological measures for assessing cognitive load. In *Proceedings* *of the 12th ACM international conference on Ubiquitous computing*, Ubicomp '10, pages 301–310, New York, NY, USA. ACM.

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

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

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

Hassenzahl, M. and Tractingsky, N. (2006). User experience – a research agenda. *Behaviour & Information Technology*, 25(2):91–97.

Hervas, R. (2009). *Context Modeling for Information Visualization in Intelligent Environments*. PhD thesis, Castilla-La Mancha University.

Hervás, R. and Bravo, J. (2011). Towards the ubiquitous visualization: Adaptive user-interfaces based on the semantic web. *Interact. Comput.*, 23(1):40–56.

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

Hinckley, K., Pierce, J., Horvitz, E., and Sinclair, M. (2005). Foreground and background interaction with sensor-enhanced mobile devices. *ACM Trans. Comput.-Hum. Interact.*, 12(1):31–52.

Ho, J. and Intille, S. S. (2005). 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, pages 909–918, New York, NY, USA. ACM.

Höök, K. (2000). Steps to take before intelligent user interfaces become real. *Interacting with Computers*, 12(4):409–426.

Horn, P. (2001). Autonomic Computing: IBM's Perspective on the State of Information Technology. Technical report.

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

Horvitz, E., Koch, P., Sarin, R., Apacible, J., and Subramani, M. (2005). Bayesphone: precomputation of context-sensitive policies for inquiry and action in mobile devices. In *Proceedings of the 10th international conference on User Modeling*, UM'05, pages 251–260, Berlin, Heidelberg. Springer-Verlag.

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

IBM (2006). An architectural blueprint for autonomic computing.

Indulska, J. and Sutton, P. (2003). Location management in pervasive systems. In *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003 - Volume 21*, ACSW Frontiers '03, pages 143–151, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Isbell, C., Shelton, C. R., Kearns, M., Singh, S., and Stone, P. (2001). A social reinforcement learning agent. In *AGENTS 2001: Autonomous agents*, pages 377–384. ACM.

Iso (2010). ISO 9241-210:2010 - Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems.

Jackson, T., Dawson, R., and Wilson, D. (2001). The cost of email interruption. *Journal of Systems and Information Technology*, 5(1):81–92.

Jaffar, J. and Maher, M. J. (1994). Constraint logic programming: a survey. *The Journal of Logic Programming*, 19–20, Supplement 1(0):503 – 581.

Jaimes, A. and Sebe, N. (2007). Multimodal human-computer interaction: A survey. *Computer Vision and Image Understanding*, 108(1-2):116 – 134. Special Issue on Vision for Human-Computer Interaction.

Jameson, A. (2002). Usability issues and methods for mobile multimodal systems. In *Proceedings of the ISCA Tutorial and Research Workshop on Multi-Modal Dialogue in Mobile Environments*, Kloster Irsee, Germany.

Ju, W. (2008). *The design of implicit interactions*. PhD thesis, Department of Mechanical Engineering, Stanford University.

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

Karagiannidis, C., Koumpis, A., and Stephanidis, C. (1997). Adaptation in immps as a decisions making process. *Computer Standards and Interfaces*, 18.

Kephart, J. and Chess, D. (2003). The vision of autonomic computing. *Computer*, 36(1):41 – 50.

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

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

Krug, S. (2005). *Don't Make Me Think: A Common Sense Approach to the Web (2nd Edition)*. New Riders Publishing, Thousand Oaks, CA, USA.

Kuniavsky, M. (2010). *Smart Things: Ubiquitous Computing User Experience Design.* Morgan Kaufmann, Burlington, MA.

Latorella, K. A. (1998). Effects of modality on interrupted flight deck performance: Implications for data link. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 42(1):87–91.

Law, E. L.-C., Roto, V., Hassenzahl, M., Vermeeren, A. P., and Kort, J. (2009). Understanding, scoping and defining user experience: a survey approach. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, pages 719–728, New York, NY, USA. ACM.

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

Lewis, J. R. (1995). Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journ. Human-Computer Interaction*, 7(1):57–78.

Limbourg, Q., Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., and Trevisan, D. (2004). Usixml: A user interface description language for context-sensitive user interfaces. In *in proceedings of the acm avi'2004 workshop developing user interfaces with xml: advances on user interface description languages*, volume 25, pages 55–62.

Littman, M., Ravi, N., Fenson, E., and Howard, R. (2004). Reinforcement learning for autonomic network repair. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 284–285.

Maeda, J. (2006). *The Laws of Simplicity.* The MIT Press.

Maes, P. (1994). Agents that reduce work and information overload. *Commun. ACM*, 37(7):30–40.

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

Mäntyjärvi, J. and Seppänen, T. (2002). Adapting applications in mobile terminals using fuzzy context information. In *Proceedings of the 4th International Symposium on Mobile Human-Computer Interaction*, Mobile HCI '02, pages 95–107, London, UK, UK. Springer-Verlag.

Mao, J.-Y., Vredenburg, K., Smith, P. W., and Carey, T. (2001). 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, pages 12–. IBM Press.

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

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

McCarthy, J. and Wright, P. (2004). Technology as experience. *interactions*, 11(5):42–43.

McCrickard, D. S. and Chewar, C. M. (2003). Attuning notification design to user goals and attention costs. *Commun. ACM*, 46(3):67–72.

Mcfarlane, D. C. (1997). Interruption of people in human-computer interaction: A general unifying definition of human interruption and taxonomy.

McFarlane, D. C. (1999). Coordinating the interruption of people in human-computer interaction. In Sasse, A. and Johnson, C., editors, *Proceedings of Human-Computer Interaction (INTERACT 1999)*, pages 295–303. IOS Press.

McFarlane, D. C. and Latorella, K. A. (2002). The scope and importance of human interruption in human-computer interaction design. *Hum.-Comput. Interact.*, 17(1):1–61.

Miller, J. and Mukerji, J. (2003). MDA Guide Version 1.0.1. Technical report, Object Management Group (OMG).

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

Mori, G., Paterno, F., and Santoro, C. (2004). Design and development of multidevice user interfaces through multiple logical descriptions. *Software Engineering, IEEE Transactions on*, 30(8):507 – 520.

Mostefaoui, G., Pasquier-Rocha, J., and Brezillon, P. (2004). Context-aware computing: a guide for the pervasive computing community. In *Pervasive Services, 2004. ICPS 2004. Proceedings. The IEEE/ACS International Conference on*, pages 39 – 48.

Mostéfaoui, G. K., Pasquier-Rocha, J., and Brézillon, P. (2004). Context-aware computing: A guide for the pervasive computing community. In *Proceedings of the IEEE/ACS International Conference on Pervasive Services (ICPS'04)*, pages 39–48.

Nagata, S. F. (2003). Multitasking and interruptions during mobile web tasks. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 47(11):1341–1345.

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

Neerincx, M., van Doorne, H., and Ruijsendaal, M. (2000). Attuning computer-supported work to human knowledge and processing capacities in ship control centres. In Schraagen, J., Chipman, S., and Shalin, V., editors, *Cognitive Task Analysis*, Mahwah and NJ. Erlbaum, Erlbaum.

Nigay, L. and Coutaz, J. (1993). A design space for multimodal systems: concurrent processing and data fusion. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 172–178, New York, NY, USA. ACM.

Norman, D. A. (2005). Human-centered design considered harmful. *interactions*, 12(4):14–19.

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

OMG (2006a). Business Process Modeling Notation (BPMN) Specification. OMG Final Adopted Specification.

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

Oulasvirta, A. and Saariluoma, P. (2004). Long-term working memory and interrupting messages in human-computer interaction. *Behav. Inf. Technol.*, 23(1):53–64.

Oviatt, S. (1999). Ten myths of multimodal interaction. *Commun. ACM*, 42(11):74–81.

O'Grady, M., O'Hare, G., and Keegan, S. (2008). Interaction modalities in mobile contexts. In Virvou, M. and Jain, L., editors, *Intelligent Interactive Systems in Knowledge-Based Environments*, volume 104 of *Studies in Computational Intelligence*, pages 89–106. Springer Berlin / Heidelberg.

Panach, J. I., Condori-Fernández, N., Baars, A., Vos, T., Romeu, I., and Pastor, O. (2011). Towards an experimental framework for measuring usability of model-driven tools. In *Proceedings of the 13th IFIP TC 13 international conference on Human-computer interaction - Vol. Part IV*, INTERACT'11, pages 640–643, Berlin, Heidelberg. Springer-Verlag.

Paramythis, A., Weibelzahl, S., and Masthoff, J. (2010). Layered evaluation of interactive adaptive systems: framework and formative methods. *User Modeling and User-Adapted Interaction*, 20(5):383–453.

Paternò, F. (2003). From model-based to natural development. *HCI International*, pages 592–596.

Paternò, F., Santoro, C., and Spano, L. D. (2012). Deliverable 2.4.2 criteria for the evaluation of caa of sfes. Technical report, Project no. FP7 - ICT - 258030.

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

Piva, S., Bonamico, C., Regazzoni, C., and Lavagetto, F. (2005). *A Flexible Architecture for Ambient Intelligence Systems Supporting Adaptive Multimodal Interaction with Users*, chapter 6, pages 97–120. IOS Press, Amsterdam.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., and Carey, T. (1994). *Human-computer interaction.* Addison-Wesley Publishing Company.

Ramchurn, S., Deitch, B., Thompson, M., De Roure, D., Jennings, N., and Luck, M. (2004). 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*, pages 364 – 371.

Reeves, L. M., Lai, J., Larson, J. A., Oviatt, S., Balaji, T. S., Buisine, S., Collings, P., Cohen, P., Kraal, B., Martin, J.-C., McTear, M., Raman, T., Stanney, K. M., Su, H., and Wang, Q. Y. (2004). Guidelines for multimodal user interface design. *Commun. ACM*, 47:57–59.

Rosenthal, S., Dey, A. K., and Veloso, M. (2011). Using decision-theoretic experience sampling to build personalized mobile phone interruption models. In *Proceedings of the 9th international conference on Pervasive computing*, Pervasive'11, pages 170–187, Berlin, Heidelberg. Springer-Verlag.

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

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

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

Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, 14(2):131–164.

Savio, N. and Braiterman, J. (2007). Design sketch: The context of mobile interaction. In *Proceedings of MobileHCI 2007*, pages 248–286.

Schilit, B., Adams, N., and Want, R. (1994). Context-aware computing applications. In *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, WMCSA '94, pages 85–90, Washington, DC, USA. IEEE Computer Society.

Schmandt, C., Marmasse, N., Marti, S., Sawhney, N., and Wheeler, S. (2000). Everywhere messaging. *IBM Systems Journal*, 39(3.4):660 –677.

Schmidt, A. (2000). Implicit human computer interaction through context. *Personal Technologies*, 4(2-3):191–199.

Schmidt, A. (2013). *Context-Aware Computing: Context-Awareness, Context-Aware User Interfaces, and Implicit Interaction.* The Interaction Design Foundation, Aarhus, Denmark.

Schmidt, A., Beigl, M., and Gellersen, H.-W. (1999). There is more to context than location. *Computers & Graphics*, 23(6):893 – 901.

Schmidt, A., Pfleging, B., Alt, F., Sahami, A., and Fitzpatrick, G. (2012). Interacting with 21st-century computers. *Pervasive Computing, IEEE*, 11(1):22 –31.

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

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

Serral, E. (2011). *Automating Routine Tasks in Smart Environments: A Context-aware Model-driven Approach.* PhD thesis, Universitat Polit'ecnica de Val'encia.

Serral, E., Valderas, P., and Pelechano, V. (2010). Towards the model driven development of context-aware pervasive systems. *Pervasive Mob. Comput.*, 6(2):254–280.

Sheridan, T. B. and Verplank, W. L. (1978). Human and computer control of undersea teleoperators. Technical report, Massachusetts Inst. of Tech. Cambridge Man-Machine Systems Lab.

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

Sottet, J.-S., Calvary, G., Favre, J.-M., Coutaz, J., Demeure, A., and Balme, L. (2006). Towards model driven engineering of plastic user interfaces. In Bruel, J.-M., editor, *Satellite Events at the MoDELS 2005 Conference*, volume 3844 of *Lecture Notes in Computer Science*, pages 191–200. Springer Berlin Heidelberg.

Strang, T. and Linnhoff-Popien, C. (2004). A context modeling survey. In *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England.*

Streefkerk, J. W., van Esch-Bussemakers, M. P., and Neerincx, M. A. (2006). Designing personal attentive user interfaces in the mobile public safety domain. *Computers in Human Behavior*, 22(4):749–770.

Sutton, R. and Barto, A. (1998a). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning).* The MIT Press.

Sutton, R. S. and Barto, A. G. (1998b). *Reinforcement Learning: An Introduction.* MIT Press.

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

Tedre, M. (2006). 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, pages 19–24, New York, NY, USA. ACM.

Tesauro, G. (2007). Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing*, 11(1):22–30.

Toninelli, A., Khushraj, D., Lassila, O., and Montanari, R. (2008). Towards socially aware mobile phones. In *Proceedings of the ISWC 2008 Workshop on Social Data on the Web (SDoW2008).*

Turk, M. and Robertson, G. (2000). Perceptual user interfaces (introduction). *Commun. ACM*, 43(3):32–34.

Unger, R. and Chandler, C. (2009). *A Project Guide to UX Design: For user experience designers in the field or in the making.* New Riders Publishing.

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

Valtonen, M., Vainio, A.-M., and Vanhala, J. (2009). Proactive and adaptive fuzzy profile control for mobile phones. In *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, pages 1 –3.

van Solingen, R. and Berghout, E. (1999). Goal/question/measures.

van Welie, M. (2001). *Task-based user interface desing*. PhD thesis, Dutch Graduate School for Information and Knowledge Systems.

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

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

Vertegaal, R. (2002). Designing attentive interfaces. In *Proceedings of the 2002 symposium on Eye tracking research & applications*, ETRA '02, pages 23–30, New York, NY, USA. ACM.

Vertegaal, R. (2003). Attentive user interfaces: Introduction. *Communications of the ACM*, 46(3):30–33.

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

Villamor, C., Willis, D., Wroblewski, L., and Rhim, J. (2010). Touch Gesture Reference Guide.

Voelter, M. (2013). *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*. dslbook.org.

Vuolle, M., Tiainen, M., Kallio, T., Vainio, T., Kulju, M., and Wigelius, H. (2008). Developing a questionnaire for measuring mobile business service experience. In *MobileHCI '08*, pages 53–62. ACM.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Mach. Learning*, 8(3-4):279–292.

Wei, Z.-G., Macwan, A. P., and Wieringa, P. A. (1998). A quantitative measure for degree of automation and its relation to system performance and mental load. *Human Factors*, 40:277–295.

Weiser, M. (1999). The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11.

Weiser, M. and Brown, J. S. (1997). The coming age of calm technolgy. In Denning, P. J. and Metcalfe, R. M., editors, *Beyond calculation*, pages 75–85. Copernicus, New York, NY, USA.

Whiteson, S. and Stone, P. (2006). Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917.

Wiering, M. A. (2002). Model-based reinforcement learning in dynamic environments. Technical Report CS-UU-2002-029, Utrecht University.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in software engineering: an introduction.* Kluwer Academic Publishers, Norwell, MA, USA.

Wood, S., Cox, R., and Cheng, P. (2006). Attention design: Eight issues to consider. *Computers in Human Behavior*, 22:588–602.

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

Zhai, S. (2003). What's in the eyes for attentive input. *Commun. ACM*, 46(3):34–39.

Emerging ubiquitous technologies such as mobile devices enable users to always be connected to the environment, making demands on the most precious resource for the user: **human attention**.

We are spending more and more time responding to the demands of machines.

The proposed approach avoids overwhelming the user's attention, creating a considerate environment that adapts ubiquitous services' interactions to the user's context. Like a considerate human...

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Miriam Gil Pascual

July, 2013