Gil Pascual, M.; Serral Asensio, E.; Valderas Aranda, PJ.; Pelechano Ferragud, V. (2013). Designing for user attention: a method for supporting unobtrusive routine tasks. Science of Computer Programming. 78(10):1987-2008. doi:10.1016/j.scico.2013.03.002.

Designing for user attention: a method for supporting unobtrusive routine tasks

Miriam Gil[a,*], Estefanía Serral[b], Pedro Valderas[a], Vicente Pelechano[a]

[a] *Centro de Investigación en Métodos de Producción de Software, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain*
[b] *Christian Doppler Laboratory "Software Engineering Integration for Flexible Automation Systems" Vienna University of Technology, Austria*

Abstract

The automation of user routine tasks is one of the most important challenges in the development of Ambient Intelligence systems. However, this automation may be annoying since some tasks may grab users attention in inappropriate situations. Since user attention is a valuable resource, task automation must behave in a considerate manner demanding user attention only when it is required. To address this issue, this work presents a systematic method for supporting the design and automation of unobtrusive routine tasks that can adjust their obtrusiveness level at runtime according to the user attentional resources and context. This method proposes to design the routine tasks that the system must carry out and how they must interact with users in terms of obtrusiveness. The method also provides a software infrastructure that makes the execution of the tasks at the appropriate obtrusiveness degree a reality. Finally, the system has been validated by means of usefulness and performance tests and a practical case study that demonstrates the correctness and applicability of our approach without compromising system performance.

Keywords: Unobtrusive routine tasks, Obtrusiveness adaptation, Ambient Intelligence, Model-based approach

1. Introduction

Ambient Intelligence (AmI) is still a vision of a future where computational power is embedded in everyday appliances and physical objects to build smart environments that are capable of freeing people to a large extent from tedious routines [1]. A routine is a set of tasks that are habitually performed when similar contexts arise [2]. Opening the blinds when users wake up, cleaning twice a week, doing the shopping when needed, lowering blinds and winding up awnings when raining, or calling the police when an intruder gets into the users' home or store. This automation is one of the most important challenges in AmI. By automating the routines that users perform, we can make their lives more comfortable, efficient, and productive, providing them with more quality of life and optimizing natural resources.

However, according to Tedre [3], a task should be carefully studied before determining what to automate and to what degree it should be automated. Some tasks cannot be completely automated since user input is required, or users may prefer to know what is happening behind the scenes. For these tasks that grab users' attention, the degree of obtrusiveness should be studied to avoid overwhelming users and enhance user satisfaction. For example, a task for lowering the blinds when it is raining could be completely automated because the user probably does not want to be asked or informed about it. However, the user may want

*Corresponding author. Tel.: +34-963877007 (Ext. 83533), Fax: +34-963877359
Email addresses: mgil@pros.upv.es (Miriam Gil), estefania.serral@tuwien.ac.at (Estefanía Serral), pvalderas@pros.upv.es (Pedro Valderas), pele@pros.upv.es (Vicente Pelechano)

to be asked for confirmation before the system does the shopping. In addition, this notification should be adapted to the context and attentional resources of the user in order not to interrupt the user by using the appropriate interaction resources. For example, if the user is alone, the notification could be presented by means of speech; however, if the user is with company, it should be made in a more subtle manner. According to the Considerate Computing vision [4], user attention is a primary and limited resource to be considered, especially in AmI systems, which promote a natural interaction between the user and the environment. User attention refers to a user's ability to attend to primary tasks, ignoring system-generated distractions [5]. Thus, routines must behave in a considerate manner, demanding user attention only when it is actually required. Each time a routine is performed, some of its tasks may require a different degree of user attention. As a consequence, different pervasive resources should be used to support each task at the appropriate obtrusiveness level (e.g., using either the ambient sound or the mobile device to inform the user that shopping has to be done). In this work, we deal with this problem by adjusting the degree in which each task requests the user's attention (e.g., the obtrusiveness level) according to user's context.

Different initiatives exist to automate user routines. On the one hand, machine-learning approaches attempt to learn from user behaviour and infer user tasks to automate them [6, 7, 8]. On the other hand, context-aware rule-based approaches [9][10] program rules that trigger the sequential execution of tasks when a certain context event is produced (e.g., switching on lights when presence is detected). However, none of these initiatives have studied the intrusiveness of task automation and how to solve it by adjusting the obtrusiveness level of the tasks. If routine tasks do not behave in a considerate manner, their execution may become a burden on users instead of a way of helping them.

This article aims at addressing the unobtrusive automation of user routines by applying model-based techniques. The main contribution of this article is a systematic method for supporting the design and automation of user routines that adapt their obtrusiveness level at runtime according to user attentional resources and context. Our method proposes to design both the tasks that must be carried out according to context and the obtrusiveness levels where they can be performed. The possible pervasive services and resources for each obtrusiveness level are defined according to the required attention. On the one hand, the task modelling is supported by a context-adaptive task model that allows designers to describe the tasks to be performed in the opportune context situation. On the other hand, we provide mechanisms for describing the obtrusiveness level required for each task according to context. Finally, a software infrastructure automates the described routines adjusting their obtrusiveness level at runtime. Specifically, our approach for adaptation consists of selecting from the available pervasive services and resources those that provide an obtrusiveness level that is as close as possible to the one desired for performing the task. To sum up, the contribution of this work is a model-based method that provides:

- A technology-independent modelling language for designing routines that are unobtrusive for users.
- A model-driven software infrastructure that automates the designed routines adapting their obtrusiveness level according to user attention and context.

The paper is organized as follows. Section 2 presents the related work. Section 3 describes an overview of our approach. Section 4 explains how user routines are described in order to be automated in the opportune context. Section 5 explains how these routines are adjusted in terms of obtrusiveness and how they must interact with users in order not to be disturbing. Section 6 presents the software infrastructure that automates the described routines. Section 7 presents an evaluation of the approach by means of performance tests and a practical case study that validates the correctness of the system obtained and its applicability in terms of user satisfaction. Finally, Section 9 discusses the approach and presents the conclusions and future work.

2. Related work

This work deals with automating user routines by taking into account the intrusive and context-aware nature of this automation. The groups that have been working on MavHome [6], CASAS [8] and iDorm [7]

are some of the most active groups in this area. They have done a great job automating the daily tasks of users by applying machine-learning approaches that learn from the past behaviour of users. A survey of the most important works in this area can be found in [11]. The main disadvantage of these approaches is that they may be intrusive for users because they do not usually take into account the users' desires (e.g., the repeated execution of an action does not imply that the user wants this automation). Moreover, they can only reproduce the actions that users have frequently executed in the past and in the same manner that the users have executed them. The work of Castanedo et al. [12] proposes a multi-agent system that follows a Belief-Desire-Intention model to achieve rational actions based on human reason in an ambient intelligent system. Other works, such as the ones presented in [9] and [10], propose programming event-condition-action rules that trigger the sequential execution of actions when a certain context event is produced. These approaches take into account context information; however, the rules are fixed hard-coded and are difficult to maintain. Overall, none of the presented approaches consider the obtrusiveness of the automated tasks or provide a high-level modelling language for describing unobtrusive routines. In contrast, we propose a technology-independent modelling language that allows the obtrusiveness aspects to be introduced in the routine automation. We also provide a software infrastructure that dynamically varies the obtrusiveness level of the automated tasks, demanding the appropriate attentional resources in each context.

With regard to interactive system adaptation to users, research efforts have been focused on dealing with modelling and adapting system interaction to different contexts of use. Approaches in this area allow the information contained in the models to be exploited for user interface adaptation in response to context changes. Calvary et al. [13] give an overview of different modelling approaches to deal with user interfaces supporting multiple targets in the field of context-aware computing. The work of Hervás et al. [14] presents a conceptual model to link context information with pervasive elements in order to personalize the offered services of these elements. Other works, such as DynaMO-AID [15] or the work of Blumendorf et al. [16], define a runtime architecture to process context data to support migration, distribution, and multi-modality. However, these works define the adaptation space in terms of the environment and the platform. Our approach defines the adaptation space for the interaction in terms of obtrusiveness for achieving seamless interactions. Thus, we address a different issue that is more related to human limitations (e.g., user attention) than device technical limitations (e.g., screen size).

In order to create systems that adapt their level of intrusiveness to the context of use, works are mainly focused on minimizing unnecessary interruptions to the user [17]. Moreover, the amount of studies concentrating on the design of unobtrusive interactions is still limited. Approaches in the area of Considerate Computing [4] are mainly focused on detecting or inferring attention by calculating the cost of interruption in order to predict acceptability. Horvitz et al. [18] demonstrated the potential use of Bayesian networks for computing the cost and value of interruptions. Hinckley and Horvitz [19] modelled interruptibility by considering the user's likelihood of response and the previous and current activity. Ho and Intille [20] suggested that proactive messages delivered when the user is transitioning between two activities may be received more positively after comparing different mental stages during which interruption occurs. Vastenburg et al. [21] conducted a user study of acceptability of notifications to find out what factors influence their acceptability. Although these initiatives recognize the need to adapt the interaction, efforts have been placed on minimizing unnecessary interruptions, overlooking automation aspects that we have taken into account in this work. In addition, we define the manner in which the interaction of each task is adapted according to context changes.

3. An overview of the proposal

The use of pervasive technologies provides new opportunities for automating user routines. However, according to Tedre [3], what to automate and to what degree it should be automated should be carefully studied. In the environments where user routines are performed, users may be physically, socially, or cognitively engaged (e.g., in a meeting, sleeping, etc.) requiring different degrees of user attention. Thus, routines should be able to dynamically adjust their obtrusiveness to the current user attentional resources and context. For example, a task of a routine that adjusts its obtrusiveness informs users about their new messages either by using the ambient sound if they are alone and can listen to them, or in the mobile device

if they are not alone and cannot be disturbed at that moment. A non-adaptive task informs the user in the same manner regardless the situation of the user. As shown in the example, depending on the pervasive resources used for performing the task of the routine (mobile device display, lamp lights, mobile vibration, TV display, etc.), the degree of user attention that a task needs varies.

To master the complexity in the development of these complex AmI systems, our approach makes use of model-based techniques. When modelling a routine, it is not possible to determine the user situation or the degree of user attention available when the task is performed. Thus, the obtrusiveness level must change during runtime. We address this issue by (1) modelling the way in which the obtrusiveness level depends on context conditions, and (2) propagating the changes in the obtrusiveness level into the AmI system components. Using model-based techniques, the following benefits are obtained:

- **Focus on the process.** Separation of concerns is promoted by our approach in order to allow designers to focus on one specific aspect of the routine tasks at a time. Designers can define the way in which the obtrusiveness level of a task varies according to context conditions without thinking about technology limitations. They can think about the way they want the routines to be executed, and, later, the appropriate pervasive resources can be chosen to cope with their obtrusiveness requirements.
- **Explore the solution space.** The use of models allows capturing not only a specific solution but also capturing the rationale behind it. In this way, alternative solutions can be reconsidered and the design knowledge can be better reused for similar problems. In addition, support for traceability allows the pervasive resources affected to be easily identified when the obtrusiveness level varies.
- **Reuse of existing tools.** 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 AmI components.

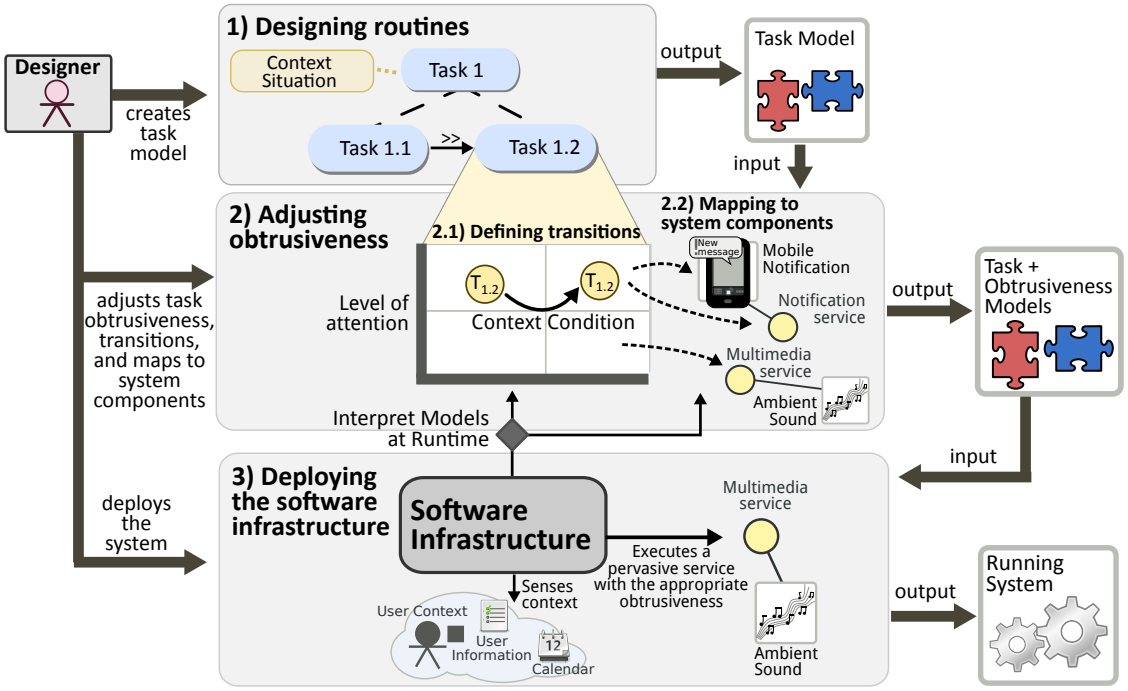

Figure 1: Proposal overview

To allow the automation of routines that adjust the obtrusiveness of their tasks according to context and user attentional resources, we provide a systematic method that leverages the descriptions of design

time to support this automation at runtime. Our method is divided into three main steps: *design of the routines*, *adjustment of the obtrusiveness*, and *deployment of the software infrastructure* (see Figure 1). In the first two steps, designers create the models that define the routines that must be automated and the obtrusiveness levels of the tasks of the routines. These models are machine-processable and precise enough to be automatically executed and can act as executable models. The executable models constitute the input of the third step. In this step, designers deploy the software infrastructure that executes the models by interpreting them at runtime. Specifically, the steps of our method are the following (see Figure 1).

1. **Designing routines.** Each routine to be automated is described as a coordination of context-adaptive tasks that are performed proactively in reaction to a context situation. To achieve this, task modelling techniques are used. They allow designers to focus on the tasks that must be performed at a high level of abstraction, deferring technological decisions and implementation issues. From this step, designers obtain a task model where the routines are designed.
2. **Adjusting the obtrusiveness.** For each task of the designed routines, the degree of obtrusiveness required for its interaction is adjusted. This is done by specifying the possible obtrusiveness levels of the actor/entity that takes the *initiative* (user or system that starts the interaction) and *attentional demand* (user awareness of the interaction) in which the task can be executed, and by defining the transitions among the levels. Changes in the obtrusiveness level must be propagated to the underlying system components in order to use the appropriate pervasive services and resources in each situation. Thus, designers must determine the system components that must be used to provide the task functionality in the different obtrusiveness levels. The result of this step is the link between the task model and the obtrusiveness model as well as the mapping between the obtrusiveness model and the system components.
3. **Deploying the software infrastructure.** Finally, designers run the system by means of a software infrastructure. This infrastructure executes the models of the previous steps to automate the designed routines at the appropriate obtrusiveness level according to the user attentional resources and context. To achieve this, the infrastructure interprets the designed models at runtime and executes the tasks of the routines using the most appropriate pervasive services and resources in terms of obtrusiveness. With this step, designers obtain the running system.

These steps are further detailed in the following sections.

4. Designing routines

Designers have to specify the routines that must be supported for each user. To capture the routines required for each user, a requirement elicitation process is applied. This process is described in [22]. In the pre-design phase, the design team makes interviews and observations that are the basis for creating these routines.

As a running example, we are going to use the following routine named *shopping* (see Figure 2): when there are enough items in the shopping list or an urgent item has been added to the list, the shopping has to be done. To do this, two options can be followed, depending on the shopping cost:

- If the cost is higher than the minimal cost so that home delivery service is free, the home delivery service is used to do the shopping automatically. If the user can receive notifications according to his/her context and attentional resources, the system asks the user for confirmation before doing the shopping, and also informs the user afterwards. The way to present these notifications will be adjusted in order not to disturb the user.

- On the contrary (when shopping delivery is not free), the user prefers to go shopping by himself/herself. To do this, the system informs the user that the shopping needs to be done. Then, the system searches for the supermarkets where the list of items to buy is cheaper. When the user does not have urgent tasks to do[1], the system shows the supermarkets found to let the user choose one, and then it shows

[1]This information is obtained from the user agenda and represented in the context model as user information.

how to arrive at the selected supermarket. Finally, when the user is there, the system reminds the user of the list of items to buy.

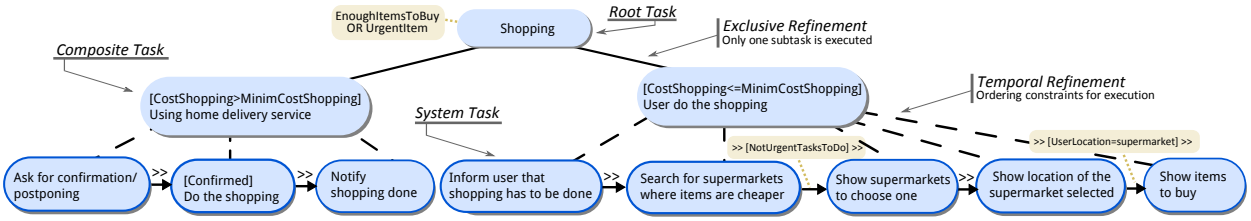

Figure 2: Routine tasks for shopping

As the definition states and the above example shows, a routine is made up of two essential parts: the set of tasks that compose the routine (i.e., ask users for confirmation, do the shopping using the home delivery service, inform users, etc.); and the context in which the routine is performed (i.e., enough items in the shopping list, user's agenda, user's location). For the specification of the routines, we propose a context-adaptive task model since it can be very expressive to describe human-computer interactions [23] and also can be specified in a machine understandable way. Using this model, designers describe each routine as a coordination of tasks that are performed in the opportune context situation.

Figure 2 shows the modelling of the *shopping* routine using the proposed task model. The root task represents the routine and has an associated context situation (*EnoughtItemsToBuy OR UrgentItem*), which defines the context conditions whose fulfillment enables the execution of the routine. This root task can be broken down into *Composite Tasks* (which are intermediate tasks) and/or *System Tasks* (which are leaf tasks). Composite Tasks are used for grouping subtasks that share a common behaviour or goal, such as the *using home delivery service* task. System Tasks represent tasks that are carried out by the system, such as *do the shopping* or *notify shopping done* tasks. For each system task, the obtrusiveness levels that can be executed are indicated as well as which pervasive services and resources must be used in each one of those obtrusiveness levels to carry out the task. The next section explains this in detail.

Also, system tasks can have input and output parameters[2]. The input parameters correspond to the parameters that the service related to the task may require to be executed. For instance, the *show location of the supermarket selected* task needs the selected supermarket as input parameter. The output parameter corresponds to the return value that the service related to the task may have. This parameter can be used as an input parameter in any of the next tasks of the routine. For instance, the *show supermarkets to choose one* task returns the chosen supermarket.

In addition, a task (both System Task and Composite Task) can have a context precondition (represented between brackets), which defines the context situation that must be fulfilled in order for a task to be performed (if the precondition is not fulfilled, the task will not be executed). In addition, composite tasks can be decomposed into simpler subtasks using *temporal* or *exclusive* refinements. With temporal refinements, all the subtasks are executed following a temporal order; in this case, tasks are related to each other using temporal operators based on the ones proposed in [24]. Specifically, the following temporal operators are provided:

- *Enablement* ($T_1 \gg T_2$): task T_2 is triggered when task T_1 finishes.
- *Task Independence* ($T_1 \mid\!=\!\mid T_2$): T_1 and T_2 can be performed in any order.
- *Enablement when the condition is fulfilled* ($T_1 \gg [c] \gg T_2$): after the completion of T_1, T_2 is started as soon as the condition c holds.
- *Enablement after t minutes* ($T_1\ t \gg T_2$): after the completion of T_1, T_2 is started as soon as the time period t has elapsed.

[2]These parameters have not been graphically included in the model in order not to overload it.

For instance, as shown in Figure 2, the *using home delivery service* task is decomposed into three subtasks using temporal refinements. This means that when the cost of the shopping is higher than the minimal cost that makes home delivery service free, the system performs the subtasks in the order specified. Since these subtasks are related by the Enablement operator, they are performed in a sequential order. On the other hand, with exclusive refinements only the first subtask whose context precondition is fulfilled will be executed. For instance, the root task of Figure 2 is decomposed into two subtasks by using exclusive refinements; therefore, to do the shopping either the home delivery service is used or the user is informed to do it.

To specify the context conditions (in the context situation, task preconditions, and relationships), we use logical expressions. Context conditions are evaluated as true or false and are composed of one or more simpler conditions that are linked together by the following logical connectives: and (AND), or (OR), equalities (=), inequalities (!=), and greater (>), or less than (<). The simplest condition is described as a logical expression composed of a left member (which has to be a context property defined on a context model), a comparative operator, and a right part (which has to be a value or another context property). In this work, an ontology-based context model implemented in the Web Ontology Language (OWL) is used to capture and reason about all the context properties that are needed for defining the context-adaptive task model. Specifically, Jena 2.4[3] and the Pellet reasoner 1.5.2.[4] are used for querying and reasoning about the model. The context properties that are used in the context conditions have to be previously defined in the OWL context model (see [25] for more detail about the context model). To refer to a context property, the name of the context property specified in the context model and the name of the instance to which this property belongs have to be indicated (e.g., a context property could be the *userLocation* property of the *Bob* instance).

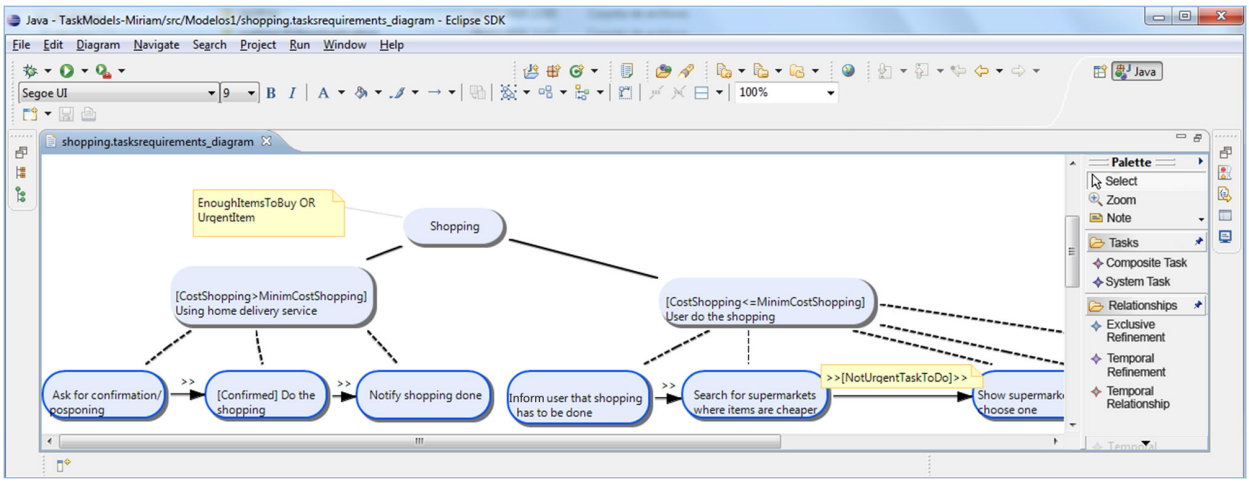

Figure 3: Snapshot of the task model modelling tool

In order to support the graphical specification of the proposed context-adaptive task model, we have developed a graphical tool, which allow task models to be defined with the notation presented above. For the implementation of the graphical tool, we have used the facilities offered by the Eclipse Graphical Modelling Framework (GMF), which is part of the Eclipse Modelling Project[5]. GMF provides a generative component and runtime infrastructure for developing graphical editors based on the Eclipse Modelling Framework (EMF). The developed tool incorporates a palette of the different components previously defined to create the task model: tasks, refinements, and relationships between tasks. In addition, this tool also provides

[3]http://jena.sourceforge.net/
[4]http://clarkparsia.com/pellet/
[5]http://www.eclipse.org/modeling/gmp/

model-based validations to ensure that the specified routines are valid prior to their construction. This allows a created task model to be automatically validated. Figure 3 shows the modelling environment.

5. Adjusting the obtrusiveness

Since user attention is a valuable but limited resource, routines must behave in a considerate manner, demanding user attention only when it is required [4]. In the execution of a routine, the user must be aware of the relevant information only. It is the designer who has to determine which tasks the system can perform automatically and which ones the user must be aware of. Thus, once routine tasks are defined, designers have to specify the way they are executed.

We make use of the conceptual framework presented in [26] to determine the obtrusiveness level of each system task in the system. This framework defines two dimensions to characterize interactions: *initiative* and *attention*. According to the *initiative* factor, interaction can be *reactive* (the user initiates the interaction) or *proactive* (the system takes the initiative). With regard to the *attention* factor, an interaction can take place at the *foreground* (the user is fully conscious of the interaction) or at the *background* (unadvised interaction). For this work, we have divided the attention axis in three segments as shown in Figure 4 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 s/he makes some effort), and *completely aware* (the user becomes aware of the interaction even if s/he is performing other tasks).

There are other frameworks for the definition of implicit interactions [27, 18]. However, the consideration of initiative and attention as independent concepts is very useful to analyze routine tasks since automation (initiative) and user awareness (attention) are factors that usually vary independently from task to task.

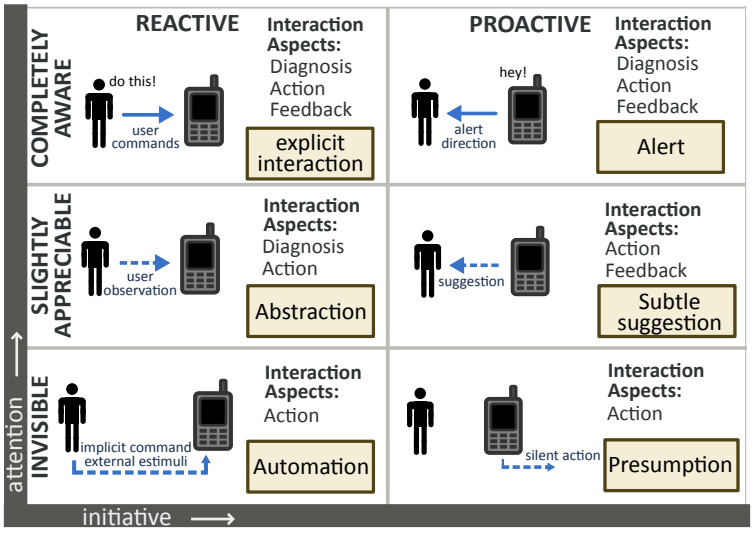

Figure 4: Conceptual framework used for the definition of implicit interactions

Figure 4 illustrates the framework and shows examples of interactions for each quadrant. These examples are:

Reactive/completely aware. Interactions take place *explicitly* by means of user commands. The user is actively engaged in diagnosing, deciding, and performing each task, and feedback is obtained. For example, if the user decides to do the shopping by means of internet, s/he performs the action, and then the system informs the user about the results obtained.

Reactive/slightly appreciable. In this quadrant, users actively perform the task of diagnosis, but the action occurs in the background of the user's attention. This is an example of *abstraction* because the user does not have to actively perform each step of the task. For example, the user is in front of a supermarket

and the system provides him/her with the shopping list in a subtle manner because the list does not contain an urgent item.

Reactive/invisible. Interactions occur in response to user actions or external stimuli without offering any feedback to the user. This interactions can help to perform routine tasks automatically with little or no user oversight (*automation*). For example, the system updates the shopping list automatically when an item runs out and the container is thrown away in a bin without first notifying the user and without providing any feedback.

Proactive/completely aware. In this quadrant, the system provides unsolicited information (*alerts*) or guides the interaction. The system identifies the problem and guides the user through the different tasks. For example, the system alerts the user that the shopping has to be done and guides the user to do it, providing feedback about the results.

Proactive/slightly appreciable. Interactions in this quadrant occur when the system proactively performs a task and informs the user about it or *suggests* the results in a subtle manner. For example, when the system informs the user that the shopping has to be done in a subtle manner (e.g., with a small icon or using vibration).

Proactive/invisible. The system anticipates what to do and performs tasks with no feedback. This interaction may hardly be considered an interaction at all, since there is no activity or awareness on the user's part. For example, the system does the shopping in an invisible manner without informing the user.

There are different ways to accomplish a task with different degrees of attentional demand and initiative. The selection of one will depend on the user attentional demand and context. Furthermore, all the tasks do not make sense in all the quadrants. For example, the task for *informing the user that shopping has to be done* makes more sense in proactive levels (e.g., the system is the one who initiates the interaction to inform the user).

Not only does the designer need to decide what action needs to occur, but also, very importantly, the manner in which it should take place by means of selecting the relevant obtrusiveness levels for each task.

5.1. Defining the transitions

During execution, tasks are performed in only one of the levels that are determined by the partitions of the obtrusiveness space. The obtrusiveness level chosen at each moment depends on the context conditions that are previously defined by the application designers. A system task will be executed in the obtrusiveness level whose context conditions are satisfied at that moment. These context conditions are constructed in the same way as the ones described in the task model.

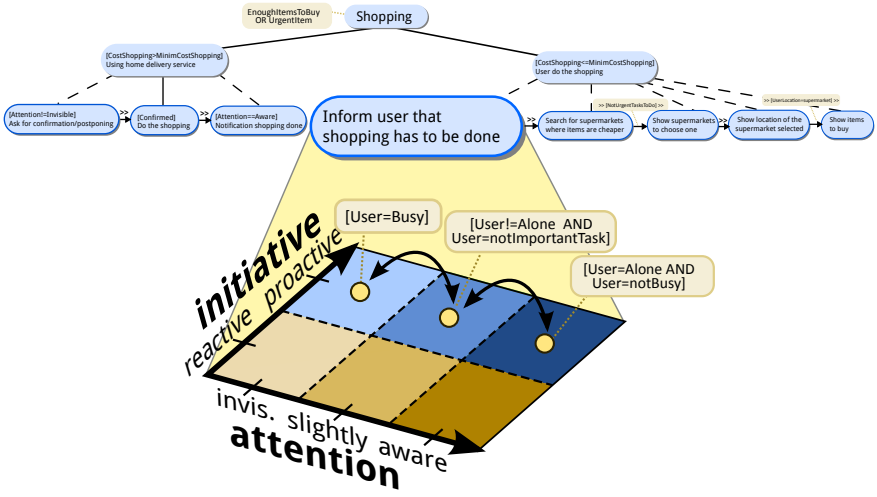

Figure 5: Selection of the obtrusiveness levels for a task

Figure 5 illustrates the linkage between the system task *inform user that shopping has to be done* and the obtrusiveness space for the interactions that support this routine. This task is in charge of *proactively* informing the user that shopping has to be done. This notification demands a different level of attention depending on the engagement of the user in other activities and whether or not s/he is alone. The notification is provided in a *notorious manner* requiring a high level of attention if the user is not engaged in other activities and is alone. The same notification is provided in a more *subtle manner* if the user is not alone and s/he is engaged in a not very important task. Otherwise, if the user is very busy and cannot be disturbed, the task is carried out in an *invisible manner* without explicitly notifying the user. In this case, the notification would be inserted in the list of received notifications without explicitly notifying the user. Thus, the user is not interrupted and can access the list of notifications to check them when s/he can.

In order to support the graphical specification of the obtrusiveness model, we have also developed a graphical tool using the possibilities offered by the Eclipse Graphical Modelling Framework (GMF). Figure 6 shows the graphical tool that support the definition of the obtrusiveness model.

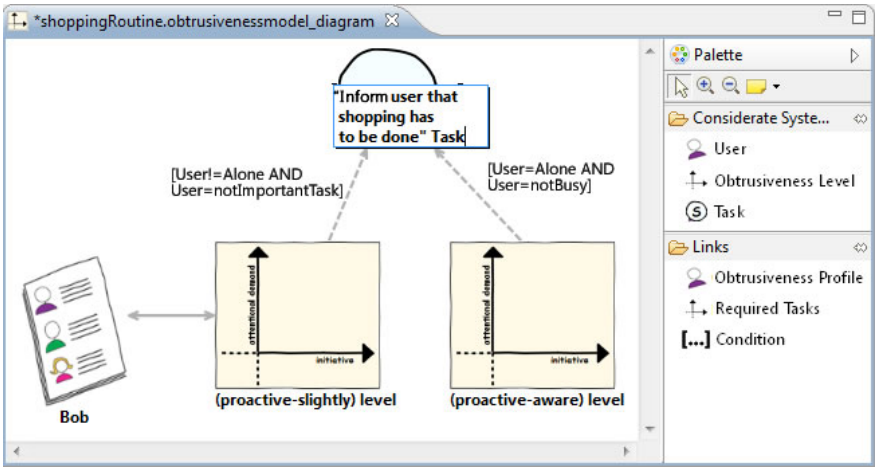

Figure 6: Snapshot of the obtrusiveness modelling tool

Once the context conditions of the obtrusiveness levels selected are defined, designers have to map the obtrusiveness levels to the appropriate pervasive services and interaction resources that support the underlying system. This is illustrated in the following subsection.

5.2. Mapping to system components

The set of transitions defined for the obtrusiveness space captures the criteria to be followed during execution. Once these criteria have been defined by means of models, we need to establish links between these models and the underlying *pervasive services* and *interaction resources* that support the system. We consider a pervasive service as a mechanism that provides a coherent set of functionality described in terms of atomic operations (or methods). These operations allow the system to control the *sensors* (such as presence detectors, temperature sensors, contact sensors, etc.), and the *interaction resources* of the environment (such as alarms, lights, blinds, ambient sound, mobile display, TV, etc.). Our approach uses these pervasive services in order to sense context changes and interact with the environment to perform the tasks of the routines. When a change is produced in the obtrusiveness level, the appropriate pervasive services and resources must be used accordingly.

Designers must indicate the pervasive services and the interaction resources that are required for supporting a task at the different obtrusiveness levels in which the task can be performed. Figure 7 illustrates the part of the pervasive system that supports the *shopping* routine from the example scenario. Pervasive services are represented by a circle, and sensor and interaction resources are represented by a square. The channels among services and resources are represented by lines. For example, the *ambient sound* of the

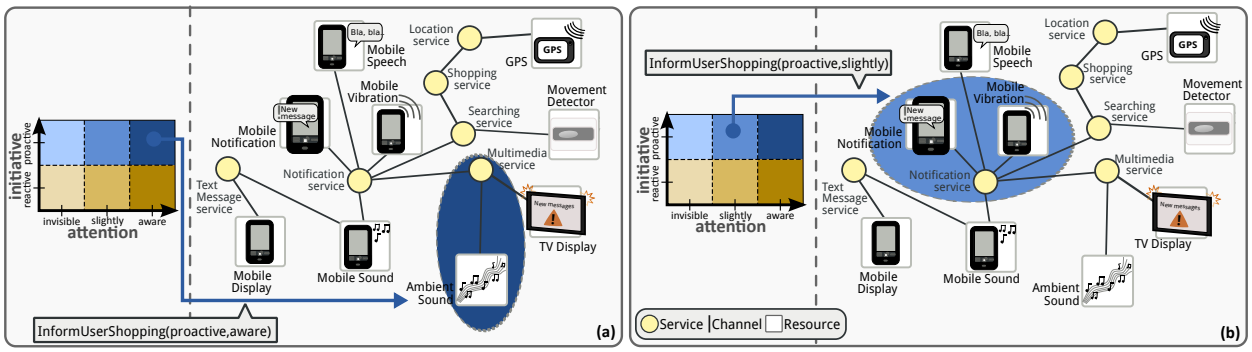

Figure 7: Mapping between obtrusiveness aspects and system components for a given task

multimedia service is used for the *inform user that shopping has to be done* task when the task is performed proactively at the highest level of attention axis. This mapping between the *ambient sound* and the obtrusiveness level for the task is expressed as *InformUserShopping(proactive,aware)* where the first coordinate refers to the initiative and the second one to the attention axis (see Figure 7a). In contrast, the *mobile notification* and the *mobile vibration* of the *notification service* are used for the same task when the task is performed at the slightly appreciable level of attention (Figure 7b).

In order to specify which pervasive services and interaction resources support a certain task for a given obtrusiveness level, the *Superimposition operator* ($\odot$) is defined. The Superimposition operator takes a task and an obtrusiveness level and returns the set of services and interaction resources required for the task. Some examples of the relationship between the obtrusiveness level for the *inform user that shopping has to be done* task and its mapping with the services and resources (see Fig. 7) are as follows:

$$\odot InformUserShopping(proactive, aware) = \{multimedia\_service, ambient\_sound\}$$
$$\odot InformUserShopping(proactive, slightlyappreciable) =$$
$$\{notification\_service, mobile\_notification, mobile\_vibration\}$$

To specify this mapping between obtrusiveness levels and pervasive services and resources, we propose the creation of a weaving model [28]. Weaving models are used to define and capture relationships between model elements by means of a set of links. Relationships between model elements are present in many different application scenarios, such as specification of transformations, traceability, or model alignment. Thus, in our approach, we use the weaving model to define the traceability between obtrusiveness levels and pervasive services and resources. Each link has the following endpoints: the first endpoint refers to tasks at an obtrusiveness level; and the second endpoint refers to system components of the pervasive system. Designers can use the ATLAS Model Weaver (AMW) tool[6] to create this weaving model.

6. Deploying the software infrastructure

In order to automate the user routines by adjusting their obtrusiveness level at runtime, we have designed and developed a software infrastructure. Thus, designers have to deploy this software infrastructure with the designed models to automate the unobtrusive routines. This infrastructure directly interprets the presented models at runtime; therefore, we reuse all the information captured in them at runtime, and we achieve that these models are the primary means to understand, interact with, and modify the routines[7].

Our infrastructure makes an intensive use of the designed models. Routines and the obtrusiveness level of their tasks are represented by the models that have been designed offline before the initial system deployment. To support the automation of the unobtrusive routines, these models are leveraged at runtime

[6] http://www.eclipse.org/gmt/amw/

[7] A video of the software infrastructure in execution is provided at http://www.pros.upv.es/routines

without modification (that is, we keep the same model representation at runtime that we use at design time). In this way, we reuse the design knowledge to achieve an unobtrusive execution of the routines. This knowledge provides us with a richer semantic base for runtime decision-making related to system execution since all the information analyzed at design time is also available at runtime.

In this section, we first explain our infrastructure and the process followed to achieve the automation of the unobtrusive routines, then we provide implementation details of it, and, finally, we illustrate how to deploy our infrastructure.

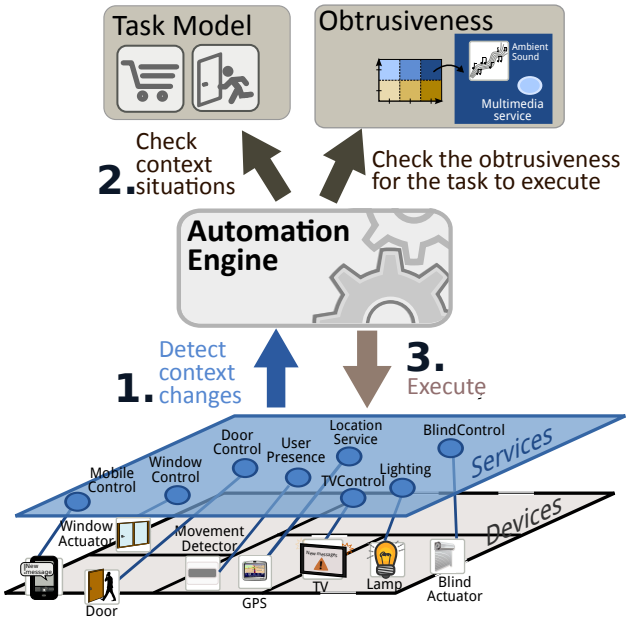

Figure 8: Software infrastructure

Figure 8 shows the components involved in the provided software infrastructure. These components are: the *design models* to be interpreted, the *automation engine* in charge of interpreting the models to automate the unobtrusive routines in the opportune context, and the *pervasive services* in charge of controlling the interaction resources of the environment. The process followed by the software infrastructure is the following:

1. **Detecting context changes:** Context changes are physically detected by sensors, which are controlled by the pervasive services provided by the smart environment, e.g., the refrigerator service controls the refrigerator. To capture context changes, the automation engine provided by the infrastructure is continuously monitoring these services. When a context change is detected, the engine updates the corresponding context properties in the context model. For instance, when the user runs out of milk, the refrigerator service detects it and the *UrgentItem* context property (see Fig. 2) is updated.
2. **Checking context situations:** When the engine detects a context change, it analyses the *Task Model* to check if there is any context situation that depends on the updated context information. For instance, when the *UrgentItem* property is updated, the engine checks whether or not there is any context situation specified in the task model that is satisfied with the new value of the property.
3. **Executing the routines:** Finally, the routines whose context situation is satisfied are executed. For instance, when the *UrgentItem* context property is set to true, the shopping routine is executed because there is an urgent item to buy. To execute a routine, the engine gets it from the task model and stores it in the dynamic memory for faster access, creating a temporal hash map structure for storing the routine parameters. The engine then executes the routine system tasks according to their refinements, their context conditions in the current context, and their temporal relationships as follows:

(a) The engine gets the first refinement of the routine:
   i. If it is an exclusive refinement, this means that only one subtask of the composite task must be performed. This subtask is the first one (from left to right) that can be executed, i.e., the first one whose context precondition is satisfied (if the task does not have a precondition, it is considered to be satisfied). Thus, the engine gets the subtask of the refinement and checks its context precondition. If it is not satisfied, the engine searches for the next refinement and repeats the step. If the precondition is satisfied, the engine executes the task. If the task is a composite task, the engine goes to step 3.a following a recursive process. If the task is a system task, the engine executes it as explained below.
   ii. If it is a temporal refinement, this means that all the subtasks must be executed in the appropriate order. Thus, the engine gets the subtask of the refinement. If its context precondition is not satisfied, the task is discarded; otherwise, the task is executed. If the task is composite, the engine goes to step 3.a following a recursive process; if the task is a system task, the engine executes it as explained below. Then, the engine checks if the task has a temporal relationship that links it to another task to be executed. If it has a temporal relationship, the engine executes the next task according to its temporal operator.

To execute a system task, the engine performs the following steps:

(a) It checks if the task has an input parameter. If so, the engine gets its value from a parameter hashmap that is created for each routine.
(b) It checks the context conditions specified in the obtrusiveness space to know what obtrusiveness level the task must be executed at. For instance, for the *inform user that shopping has to be done* task, when the user is not busy and is alone, the task must be carried out in the *proactive-completely aware* obtrusiveness level (see Figure 5).
(c) It gets the appropriate pervasive services and interaction resources to be used for that obtrusiveness level. This is done through the mapping between the obtrusiveness levels and system components (see Section 5.2). Using the superimposition operator, we easily get the set of services and interaction resources required for a task at an obtrusiveness level. For instance, for the *inform user that shopping has to be done* task the *ambient sound* of the *multimedia service* is used in the *proactive-completely aware* obtrusiveness level (see Figure 7).
(d) It executes the pervasive service with the interaction resources obtained. For instance, the engine executes the *multimedia service* with the *ambient sound* resource to carry out the *inform user that shopping has to be done* task in *proactive-completely aware* obtrusiveness level.
(e) If the task has an output parameter, the engine stores it in the parameter hashmap of the routine.

6.1. Implementation details

This section provides details about the main components of the infrastructure that supports our approach: the pervasive services and the automation engine.

1. *Pervasive Services.* We consider a service as a mechanism that provides a coherent set of functionality described in terms of atomic operations (or methods). These operations allow the system to control the devices of the environment in order to sense context and/or change it. For instance, a pervasive service could be the *shopping* service. It provides the methods: *getShoppingList*, which senses or captures the list of products that should be bought; and *doTheShopping*, which sends this list to the shop so that the products are delivered to the user's home and afterwards updates the list. Our approach uses these pervasive services in order to perform the tasks of the routines specified in the models and to sense context changes. The pervasive services are developed in Java/OSGi [8] technology. To develop those for controlling the devices connected by the pervasive network, we use a development method presented in [25], which allows us to automatically generate Java/OSGi pervasive services from high-level abstraction models.

[8] http://www.osgi.org/

The rest of the pervasive services were manually developed. For instance, to control the mobile device, we developed a web service that is based on Android[9] and the Restlet Framework[10]. This server listens to the mobile client requests and replies to them. It has been developed as an android service and to be used by the OSGi server as a web service. When it receives the interaction resources to use, this service translates them into the concrete interaction components of the underlying technology.

2. *Automation Engine.* The Automation Engine is in charge of automating the routines in a non-intrusive way. This engine is also implemented in Java/OSGi technology. Using OSGi, the engine can listen to the changes produced in the services in order to detect context changes. Context is managed by using the ontology-based context model presented in [25]. This model is represented in Ontology Web Language (OWL)[11]. OWL is an ontology markup language that greatly facilitates knowledge automated reasoning and is a W3C standard. To manage this model, we use SPARQL, which is a W3C recommendation query language.
When the engine detects a context change, it queries the task model to check which routines must be executed in the new context. Both the task model and the obtrusiveness model are represented in XML Metadata Interchange standard (XMI)[12]. To query these models at runtime, the engine uses the Eclipse Modelling Framework Model Query (EMFMQ)[13] plugin. For instance, the following expression shows the EMFMQ query to get the obtrusiveness levels described for a task.

```
new SELECT(
    new FROM(resource.getContents()),
    new WHERE(new EObjectAttributeValueCondition(
        ObtrusivenessModelPackage.eINSTANCE.getObtrusivenessLevels_TaskID(),
        new StringValue(taskID))));
```

The engine then executes the corresponding routines by carrying out their system tasks. To execute a task, the engine searches for the pervasive service associated to the task in the OSGi server by using its service registry. Then, the engine executes the service by using the Java Reflection capabilities, passing the resources that must be used as parameters.

6.2. Deployment of the infrastructure

To deploy the software infrastructure, designers have to install the automation engine and the pervasive services in an OSGi server. In addition, the models must be saved in the folder where OSGi is installed. To run the system, designers have to start the components installed in OSGi. From this moment, the context is continuously monitored by the automation engine that reflects the context changes in the context model. In this way, the engine can check if the context situation of any routine is satisfied in the new context. If so, the engine is in charge of executing these routines by interpreting the models at runtime. It is worth noting that the models are not translated to code. The automation engine interprets the models and executes the opportune pervasive resources according to context. Thus, the models are the only representation of the routines to be automated. This facilitates their understanding and maintenance.

7. Validation of the proposal

This work has been validated from different perspectives to evaluate the technology-independent modelling language for designing unobtrusive routines and the provided software infrastructure that automates the designed routines. With regard to the modelling language, we have evaluated its **usefulness** for designing and developing unobtrusive routines and the **complexity reduction** of the adaptation specification.

[9] http://www.android.com/
[10] http://www.restlet.org/
[11] http://www.w3.org/TR/owl-features/
[12] http://www.omg.org/spec/XMI/
[13] http://www.eclipse.org/modeling/emf/

With regard to the software infrastructure, we have tested its **scalability** in order to analyze to what extent system performance could be affected by interpreting complex models at runtime, and we have tested its **correctness**. We have also evaluated the **applicability** of the obtained system in terms of user satisfaction. The first and last evaluation were based on a case study and were performed with the participation of human subjects by means of a Smart Home case study.

To make these validations, we used an experimental setup that included: a scale environment with KNX devices[14], a PC running Equinox, and an HTC Magic mobile device running Android Operating System. The PC had the the following features: Pentium 4, 3.0 GHz processor and 2 GB RAM with Windows XP Professional Edition SP3 and Java 1.5 installed.

The context processing and task obtrusiveness processing is done at the server side on the PC. When a notification has to be sent to the mobile device, our system makes use of push notifications (remote notifications). The notifications processed are pushed to a notification app on the mobile phone (via the Push Notification services) when there are notifications to see. The notification 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 (C2DM) for the Android platform.

7.1. Usefulness of our design method

In this evaluation, we explain the experiment design that we used to evaluate our design method to show its usefulness in the development of unobtrusive routines from requirements to deployment. The aim of the experiment was to compare the usefulness measure obtained by our proposed method (model-driven development) over the traditional development (hand-coding development). To do this evaluation, we followed the guidelines presented by Kitchenham et al. in [29] and Wohlin et al. in [30]. In the following subsections, we present each experimental element.

7.1.1. Objectives

According to the Goal/Question/Metric template [31] the objective of the experiment was to:

Analyse	our design method
For the purpose of	evaluating its usefulness
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 hypotheses were derived:

Null hypothesis 1, H_{10}: The usefulness of our method for designing unobtrusive routines is the same as the traditional design.

Alternative hypothesis 1, H_{11}: The usefulness of our method for designing unobtrusive routines is greater than the traditional design.

7.1.2. Identification of variables

We identified two types of variables:

- **Dependent variables:** Variables that correspond to the outcomes of the experiment. In this work, usefulness was the target of the study, which was measured in terms of the following software quality factors: learning time; development time; deployment time; correcting errors time; and maintenance and evolution time [32].
- **Independent variables:** Variables that affect the dependent variables. The development method was identified as a factor that affects the dependent variable. This variable had two alternatives: (1) model-driven development (MDD) and (2) traditional development.

[14] http://www.knx.org

7.1.3. 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 modelling tools; however, they did not have experience in task modelling. Some subjects had knowledge about OSGi technology.

Objects of study. The experiment was conducted using a case study similar to the running example used throughout the paper, i.e., the routine to do the shopping (see Fig. 2). 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 routine to guide the development of the *shopping* routine. Specifically, we provided them with a traditional implementation of a *waking up* routine as well as its modelling using our method. This routine gives support to the tasks that users perform when they wake up, tasks such as: switch off the alarm, switch on the light, raise the blinds, check the weather, turn on the bathroom heating and notify the new messages. In addition, we provided the subjects with the drivers for the KNX devices and services that they needed for developing the routines.

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

- **A demographic questionnaire:** a set of questions to know the level of the users' experience in Java/OSGi programming, modelling tools, and task modelling.
- **Work description:** the description of the work/activities that the subjects should carry out in the experiment. These activities were the following: (1) the development of the shopping routine using our model-driven method and the traditional one; and (2) the modification of the *waking up* routine (adding a new task, changing the order of the tasks, and changing the obtrusiveness level of two tasks).
- **A form:** a form was defined to capture the start and completion times of each activity. Some space was left after the completion time of each activity for additional comments of the subjects about the performed activity. If some activities were performed with interruptions, subjects wrote down the times every time they started and stopped carrying out the activity; thus, the total time was derived using these start and completion times.

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 modelling techniques. This threat was also minimized by providing the subjects with the *waking up* routine, which helped and guided them in the development of the shopping routine. 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.

Further details about the objects of study and instrumentation can be found in our web page [33].

7.1.4. Experimental design and procedure

We follow 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 [30]. 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 short presentation in which general information and instructions were given. Next, the experiment started with a demographic questionnaire to capture the user's backgrounds. Afterwards, the work description and the form were given to the subjects and they started to develop the *shopping* routine following the two kinds of development (MDD and traditional) in the indicated order to each user. For each activity of the development, they filled in a form to capture the development times. Once users developed the *shopping* routine, they started to change the *waking up* routine to evaluate the maintenance and evolution. For these activities, they also filled in the form to capture the maintenance time. Specifically, the activities carried out in each part were the following:

Traditional development. Prior to implementing the case study, we provided the subjects with the necessary tutorials and tools to learn the basics of the OSGi technology needed to develop the case study following the traditional development. Then, from the case study description, they started the implementation of the *shopping* routine taking the *waking up* routine as example. Generally, they implemented the classes to support the routine functionality with the different obtrusiveness levels and the context management (sense the context from the devices and check context conditions). Then, the subjects deployed the system in the OSGi server. Once they achieved the execution of the code, they spent some time detecting and correcting code errors. Finally, we provided a set of requirement changes for the *waking up* routine in order to evaluate the maintenance and evolution. In this activity, the subjects changed the provided implementation to support the new requirements. Then, the subjects deployed the changed routine and corrected the errors.

Model-driven development. Prior to implementing the case study, we provided the subjects with a tutorial where the modelling language and the provided tools were explained. Following the model-driven development (see Section 3), the subjects first designed the *shopping* routine according to the case study description. Then, they specified the obtrusiveness of each task by using the obtrusiveness modelling tool, linking each task in a specific obtrusiveness level with the appropriate pervasive service or interaction resource. Then, the subjects saved the created models in the specific folder of the OSGi server where the software infrastructure was deployed. Once they achieved the execution of the models, they detected and corrected some modelling errors. Finally, the subjects applied the required changes in the design of the *waking up* routine. To perform the changes, they opened the provided models with the corresponding tools and performed the changes accordingly with the new requirements.

7.1.5. Analysis and interpretation of results

In this subsection, we analyse and compare the obtained results regarding the measures established in Section 7.1.2. Specifically, the usefulness has been studied based on the learning, development, deployment, correcting errors, and maintenance time. The results have been studied based on a time mean comparison and the standard deviation. Table 1 presents the descriptive statistics for each of the studied quality factors and Figure 9 shows the means comparison of the obtained measures for each method.

Quality factor	Dev. method	Mean	N	Std. deviation
Learning Time	Traditional	15,12	8	2,78
Learning Time	MDD	8,00	8	1,71
Development Time	Traditional	4,12	8	0,55
Development Time	MDD	1,21	8	0,48
Deployment Time	Traditional	0,83	8	0,62
Deployment Time	MDD	0,20	8	0,13
Correcting Errors Time	Traditional	4,27	8	1,08
Correcting Errors Time	MDD	0,31	8	0,10
Maintenance Time	Traditional	3,00	8	0,73
Maintenance Time	MDD	0,20	8	0,06

Table 1: Descriptive statistics for each quality factor

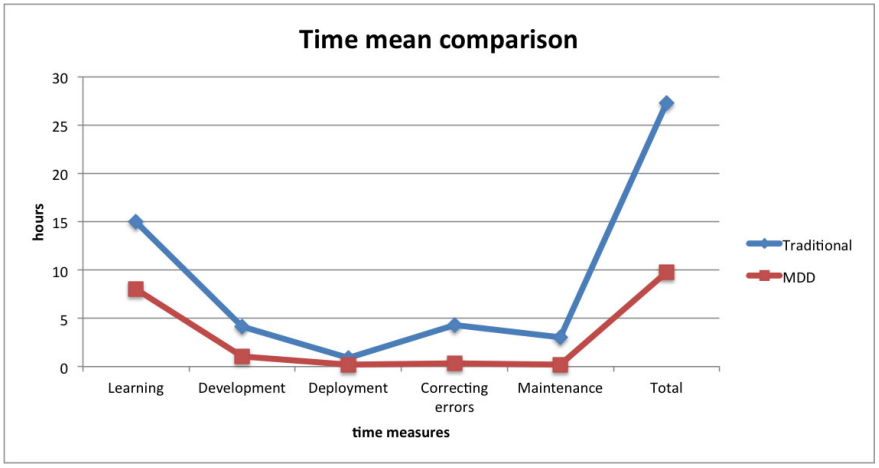

Figure 9: Time mean comparison for development method usefulness

Next, we provide a further analysis of the results for each measured software quality factor:

- **Learning time.** The learning activity in the traditional development took users between 12 and 19 hours. The subjects commented to us that having an implemented routine example (*waking up* routine) helped them in this learning activity. Nonetheless, they needed to continue learning throughout the development of the case study. This activity following the MDD method took the subjects from 7 to 9 hours. Having the *waking up* routine modelling as an example also helped them to understand the modelling language. Thus, the learning time was significantly higher in the traditional development. This is because technologies like OSGi are complicated and learning them takes longer than learning our modelling language. The MDD method provides technology-independent, high-level primitives to specify the system. Also, more dispersion was found in the learning time following the traditional development (std. deviation=2,78) because subjects that had knowledge about OSGi took much less

time than subjects without this background. It is worth noting that this activity was performed with interruptions since it took for some subjects a significant amount of time.

- **Development time.** The development time following the traditional development differed according to the users implementation experience, ranging from 3.25 (the most experienced subject) to 5 hours. Following the MDD method, the development activity ranged from 75 min to 2.20 hours. Specifically, the routine modelling took the subjects from 30 minutes in the best case to 65 minutes in the worst case. We observed that some subjects had difficulties identifying some temporal relationships between tasks because they had no experience in task modelling. However, they finally modelled the routine correctly. The obtrusiveness modelling took a little more time for users than the task modelling (from 45 minutes to 75 minutes). This was because users were less familiarized with obtrusiveness requirements than with task hierarchies. The difference between the two methods was higher since developing the routine traditionally was more complex and difficult for subjects (because they had to hard-code all the routine and obtrusiveness behaviour manually). The MDD method allowed subjects to focus on satisfying routine requirements instead of solving technological problems. Note that by following the MDD method, none of subjects had to implement anything to manage the context information since it is automatically managed by our software infrastructure and the context ontology. Regarding the standard deviation, it was low for both development methods (see Table 1) indicating that development times tended to be close for each development method.

- **Deployment time.** In this activity, the subjects deployed the system in the OSGi server. In the traditional development, the subjects that had already worked in OSGi deployed the routines in a few minutes (from 5 to 10 minutes). However, for the rest of subjects, the deployment took more time (from 40 to 70 minutes) since they did not know how to package the code into bundles and the correct order to start them in the OSGi server. For the development following the MDD method, the deployment of the system was simply required saving the models and pressing the run button. For this reason, the standard deviation was very low (0,13). Conversely, it was a little bit high for the traditional development (0,62), since some subjects deployed the system in a few minutes, and for others, it took more time.

- **Correct errors time.** In the traditional development, the subjects spent from 3 to 6 hours to detect and correct errors in the code. The common errors were having infinite loops, nested if-else with the wrong condition to specify obtrusiveness, or wrong conditions in the temporal operators of a task. For example, one subject had specified starting the *show supermarkets to choose one* task when the condition *UrgentTasksToDo* was fulfilled. The right condition was *NotUrgentTasksToDo* but it took him a lot of time to detect this error. Conversely, with the MDD method, the subjects spent from 15 to 30 minutes to detect and correct some modelling errors. Three subjects had errors in the context conditions of the obtrusiveness levels that let the system be in the same obtrusiveness level all the time. Another subject committed an error in the precondition of the *user do the shopping* task writing <instead of <=. Also, four subjects had errors in the names of the context properties used in the conditions. This was because the names have to be the same as the names specified in the context ontology. However, most of the subjects commented that the correction of faults and the performance of changes in the routines could be carried out more easily using the graphical models than analysing the code, since they could intuitively locate the things to change. The obtained standard deviation shows more dispersion following the traditional development (see Table 1), indicating that some subjects had more difficulties than others for correcting the errors.

- **Maintenance time.** With regard to the traditional development, most of the users commented that it was not easy to modify the code that they had not implemented. This activity took them from 2 to 4 hours. Changing the models following the MDD method was easier and quicker for the subjects since they already knew the semantics of the models after developing the shopping routine. This activity took less than 15 minutes for all the subjects (very low std. deviation obtained). This is because updating the routine (such as adding, modifying or deleting tasks or context information) is as easy as modifying the specified routine in the graphical tool to make it fit the new requirements.

With the MDD method, the subjects took 9.55 hours to develop the case study, whereas with the traditional development the subjects took 27.32 hours. Therefore, the development of unobtrusive routines using the MDD method (our approach) is faster than using the traditional one. In order to verify whether or not we can accept the null hypothesis, we performed a statistical study called *paired T-test* using the IBM SPSS Statistics V20 at a confidence level of 95% (α=0.05) [34]. This test is a statistical procedure that is used to make a paired comparison of two sample means, i.e., to see if the means of these two samples differ from one another. For our study, this test examines the difference in mean times for every subject with the different methods to test whether the means of the traditional development are equal to the means of the development following the MDD method. When the critical level (the significance) is higher than 0.05, we can accept the null hypothesis because means are not statistically significantly different. For our experiment, the significance of the paired T-test for the total time means is 0.000 (calculated using the IBM SPSS Statistics), which means that we can reject the null hypothesis H_{10} (the usefulness when using our design method for designing unobtrusive routines is the same as when using the traditional design). Based on this test, we have given strong evidence that the kind of development influences the usefulness. Specifically, the usefulness using our method (MDD method) is significantly better than using the traditional one, i.e., the mean values for all the measures are lower when using the MDD method; thus, the alternative hypothesis H_{11} is fulfilled: **the usefulness when using our design method for designing unobtrusive routines is greater than when using the traditional design**.

7.2. Complexity reduction of the adaptation specification

One of the main expected benefits in our approach is to reduce the complexity for specifying the unobtrusive routines adaptation by means of the technology-independent modelling language introduced. A benefit of the use of a model-driven approach is the potential reuse at modelling time. This reuse is greater with the use of ontologies [35] since ontologies provide a uniform way for specifying the core concepts of the models as well as an arbitrary number of subconcepts and facts, altogether enabling contextual knowledge sharing and reuse in an AmI system [36]. Therefore, our approach allows systematic reuse of contextual knowledge, context-adaptive tasks, and obtrusiveness levels between case studies. In addition, the support of graphical tools considerably facilitates the specification of the models and the management of their complexity by taking each aspect (context, routines, and obtrusiveness levels) as a separate concern.

The complexity of specifying an adaptive system resides in the definition of the adaptation policies, which state the adaptation space. In our system, the adaptation is defined by the context conditions specified for each possible obtrusiveness level. In order to validate the complexity reduction, we compare our modelling approach with the use of a state machine, which is usually used to represent adaptation specifications [37]. In this state machine, states represent the possible obtrusiveness levels, and transitions represent the adaptation paths between levels. The experiment is based on the obtrusiveness space of the case study (see Fig. 5). Figure 10 illustrates the state machine generated from the specification of Fig. 5 with three context conditions (three possible obtrusiveness levels). It can be observed that the resulting state machine has the complete set of adaptation paths among the states (6 adaptation paths). If we add one more obtrusiveness level to transit, the adaptation paths grow to 12. In fact, the number of adaptation paths grows rapidly as the number of obtrusiveness levels increases ($obtrusivenesslevels * (obtrusivenesslevels - 1)$), making the adaptation more difficult to specify (see Table 2). As opposed to extensionally describing each possible transition, our solution decreases complexity: adaptation is described by context conditions in each possible obtrusiveness level. Therefore, there is no need for exhaustive definitions of the adaptation paths (i.e., transitions).

7.3. Scalability of the software infrastructure

Model-based techniques provide a greater degree of flexibility for supporting the execution of unobtrusive user routines. 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*, and (b) *the model population*. Since the scalability of AmI

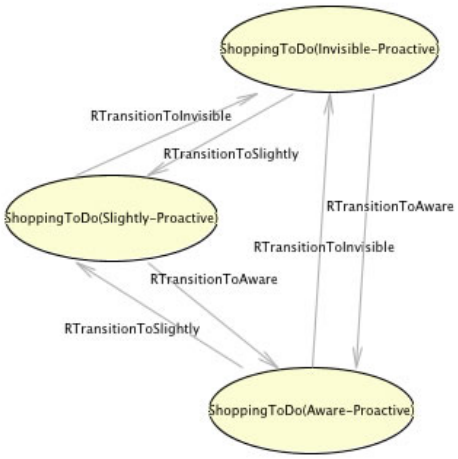

Figure 10: State machine for the adaptation space of the task to inform the user that shopping has to be done

Obtrusiveness levels	Num. Conditions (Obtrusiveness Model)	Adaptation paths (State Machine)
3	3	6
4	4	12
6	6	30

Table 2: Growing number of adaptation paths as the number of obtrusiveness levels increases in the case study

applications is an important problem rarely addressed [38], we have evaluated this concern in our software infrastructure.

When evaluating the running example of the Smart Home, the performance penalization introduced by model processing was not significant. However, in order to validate whether our proposal scales to large systems, we quantified this overhead for large models that were randomly generated. We used an empty task model and an empty obtrusiveness model to be randomly populated by means of an iterative process. The obtrusiveness model was populated with one hundred new elements each iteration, while the task model was populated with one new routine whose task structure formed a binary tree, varying its depth and the width of its first level each iteration.

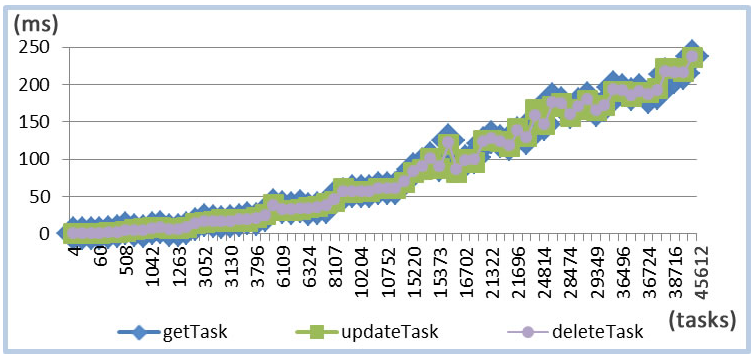

Figure 11: Temporal cost of task model operations

After every iteration, we tested all the model operations of the *Automation Engine* 20 times and calcu-

lated the average temporal cost of each one. The operation over the obtrusiveness model with the highest temporal cost was the operation to check the context conditions for choosing the appropriate obtrusiveness level. Even with an obtrusiveness model population of 45.000 elements, the model operation provided a fast response time (<450 ms) at least for the kind of conditions we are addressing. Figure 11 shows the temporal cost of the operations that access the task model with the highest cost. These operations are: getting, updating, and deleting a task. Their costs were very similar since all of them make the same query to obtain the corresponding task. Overall, even with a model population of 45612 tasks, these model operations provided a response time (250 milliseconds) that can be considered fast in the Smart Home domain that we are addressing.

It turns out that our approach gathers the necessary knowledge from the runtime models without drastically affecting the system response. 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.

7.4. Correctness and applicability of our proposal

To validate our proposal using a case study-based evaluation, we followed the research methodology practices provided by Runeson and Host [39]. These practices thoroughly describe how to conduct and report case studies. By using these practices and following our design method, we have designed and developed a smart home case study whose overall purpose is to make inhabitants stay comfortable and save energy consumption.

This case study describes several domestic scenarios for a user named Bob. In these scenarios, several routines take place, adapting their obtrusiveness according to context in order to automate user tasks in a non-intrusive way. The running example used throughout the paper makes up part of one of these scenarios. It is described as follows: Bob lives in a smart home. Every working day, the *waking up* routine is activated to get Bob up at 7 a.m. with his preferred music, light the room gradually, and turn on the bathroom heating so that it is at the suitable temperature when he takes a shower. Afterwards, Bob drinks coffee with milk for breakfast while he is watching his favourite TV program before going to work. One day during breakfast, Bob ran out of milk. In reaction to this, the refrigerator automatically added this item to the shopping list in an invisible manner for Bob and the *shopping* routine was activated because milk was an urgent item. Since the cost of the shopping was low, Bob was informed that the shopping had to be done. While he was watching the TV program, the system also reminded him that he had an important meeting at work and he had to leave the house sooner. Therefore, the recording routine was activated to start to record the program. In the afternoon, when he left the work, the mobile device showed him the supermarkets that were on the way home where the items to buy were cheaper. Bob selected the one where he preferred to shop and its coordinates were sent to his GPS to show its location. When he was inside the supermarket, the mobile device showed him the list of items to buy. When Bob arrived home, the house was the right temperature and the lights were lighting with his presence, since the outside light was already too low to raise the blinds.

With this case study, we wanted to validate that our method allows routines to be automated properly in an unobtrusive way. In particular, we wanted to evaluate whether the routines were automated adequately and whether the users were satisfied with the resulting system, thereby achieving a greater user experience level. To do this, we first developed a prototype version for the described system by following the method steps. We then performed the following two evaluations

1. *Correctness* of the software infrastructure: it validates that our infrastructure executes the designed routine tasks in the correct order and with the correct obtrusiveness level as described in the models.
2. *Applicability* of the unobtrusive automation of routines achieved in terms of user satisfaction: it validates the applicability of the system obtained by asking users whether or not they considered the execution of the routines to be appropriate in terms of obtrusiveness, taking into account the defined context events.

7.4.1. Correctness of the software infrastructure

In order to evaluate the *correctness* of our software infrastructure, we need to determine whether the system components required by each routine task are all executed in the correct order, with the correct obtrusiveness level and under the correct conditions. In the same way, we need to determine whether the routines are executed if and only if the proper context situation is fulfilled.

Historically, software engineers have used code-level tracing to capture a running system's behaviour. An alternative is to generate and analyze model-based traces, which contain rich semantic information about the system's runs at the abstraction level that its design models define. Since our *Automation Engine* enables an analysis of the overall running of the system by means of both debugging and execution trace capabilities, the proposed validation consist in: (1) simulating the fulfillment of specific context conditions in order to trigger the execution of several routines; and (2) checking that the *Automation Engine* registers the execution of all the system components that must be used to execute the tasks of the routines, in the correct order and with the correct obtrusiveness level (*correctness*).

Previous to this validation, we checked that the *Automation Engine* properly registered routine execution. Given the semantics of the design models, an engineer can check if a model-based trace is consistent with regard to a concrete run, and, more generally, if it is feasible.

To conduct the experiment, we developed a set of JUnit[15] tests that allowed us to evaluate the *correctness* of the system obtained. As a representative example, Figure 12 shows the JUnit method that evaluates the *correctness* of a routine execution. To perform this evaluation, we obtain the execution plan derived from the system tasks of the routine according to the current state of the system. To do this, we use the *getExecutionPlan()* method that constitutes one of the EMF facilities provided by the *Automation Engine*. Then, we execute the routine and retrieve the last registered automated operations of the traces registered. We retrieve as many automated operations as tasks contained in the previous plan. Finally, we create an equal assertion to check whether or not the automated operations retrieved are the same as the executed tasks that the plan contains.

```
public void TestExecuteRoutine(Routine routine){

  List<Task> executionPlan= routine.getExecutionPlan();

  executeRoutine(routine);

  List<AutomaticOperation> automatedOperations= contextModel.getLastAutomaticOperation
                                                (executionPlan.size());

  ArrayList<String> plannedTasks, executedTasks;
  for(Task t: executionPlan) planedTasks.add(t.getID());
  for(Operation o: automatedOperations) executedTasks.add(o.executedOperation.getID());

  assertEquals(plannedTasks, executedTasks);
}
```

Figure 12: JUnit test for evaluating the correctness of a routine execution

We performed these evaluations in an iterative way, which allowed us to detect and resolve some mistakes. For example, we realized that the routines that are dependent on time made the system enter into a loop. This was because the system updates time every second and the smallest time unit considered in the routines was minutes. Consequently, the context situation of these routines was continuously fulfilled until a minute passed. To solve this problem, we needed to use the same time unit in both cases. Since updating the context model every second could overload the system, we updated the *Automation Engine* so that the time was updated every minute. Thus, our research results show that the infrastructure is capable of automating the unobtrusive routines correctly as described in the models.

[15] http://www.junit.org/

7.4.2. Applicability of the approach in terms of user satisfaction

To evaluate the applicability of the approach in terms of user satisfaction, we conducted an experiment[16] in which users performed the scenario of the case study described, adopting Bob's role. In the experiment, we simulated the different environments in which the scenario takes place. Users were given a script to follow with the tasks they had to perform in order to activate the routines at a specific obtrusiveness level. The script described tasks such as waking up at a specific time, pouring a cup of milk for breakfast, finishing the milk that was left in the carton and throwing it into the trash, having breakfast while watching TV, leaving home and going to work, etc. These tasks are the ones described in the case study scenario.

In addition, in the scenario described, the tasks of the routines are presented at different obtrusiveness levels according to the user context. For example, the video recorder task is presented in a *reactive-invisible* manner because it begins to record automatically in reaction to the user leaving. However, informing the user that the shopping has to be done is presented in a *completely aware* manner because the user is not busy and is alone. In this way, the users could evaluate the applicability of the method by means of the execution of the routines and their obtrusiveness.

A total of 15 subjects between 23 and 54 years old participated in the experiment (7 female and 8 male). Most of them had a strong background in computer science, being students or researchers (10 subjects) in this field. The rest of them (5) were subjects from other areas. To evaluate the applicability of the system in terms of the user experience, we used an adapted IBM Post-Study questionnaire (PSSUQ) [40]. PSSUQ is a 19-item instrument for assessing user satisfaction with system usability. Specifically, it allows us to measure the overall satisfaction (OVERALL) of the system, its usefulness (SYSUSE), its information quality (INFOQUAL), and its interface quality (INTERQUAL). We also extended this questionnaire to include questions regarding the execution of the routines and their interaction obtrusiveness. Thus, two dimensions were evaluated: *user satisfaction* and *routine automation*. We applied a Likert scale (from 1 to 5 points) to evaluate the items of the questionnaire. Some space was left at the end of the questionnaire for further comments.

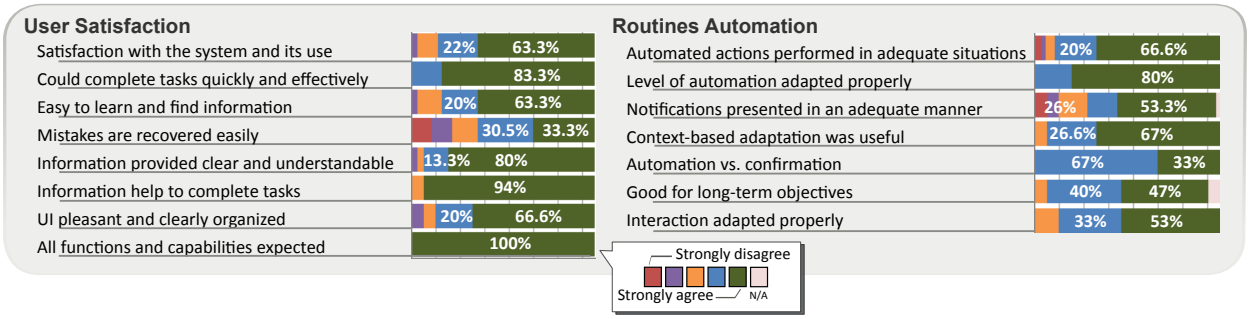

Figure 13: Summarized results

Overall, the experiment showed that by following our method, user routines can be automated by adjusting their obtrusiveness level according to context and obtaining a high degree of user satisfaction. Figure 13 shows a summarized table of the results obtained. Most of the users (85%) were satisfied with the system and how to use it. More than 80% of the people also strongly agreed that by using the system, they were able to complete the tasks and scenarios effectively and quickly. Although most users (83%) thought that it was easy to learn to use the system and to find the information needed, they complained about how to recover from mistakes. The results were also positive regarding the quality of the information provided.

In the results, the clarity and understandability of the different messages were considered to be appropriate by 80% of the subjects, and these messages helped 94% of the users to complete tasks. More than 85% of the users agreed or strongly agreed that the interfaces were pleasant and clearly organized, and all of the subjects considered that the system had all the functions and capabilities they expected for the scenario.

[16]Screenshots in http://www.pros.upv.es/routines

With regard to the automation of the routines, 86% of subjects strongly agreed or agreed in that automated actions were performed in appropriate situations interacting with them only when needed. They said that this helped them to perform the routine tasks described in the scenario. In addition, most users agreed that the level of automation was adequate and the system generally asked them for confirmation when it was essential. However, some users complained because they would like to be notified about some tasks (26%) or personalize them; for instance, they said that they would prefer the system ask them whether or not they want to use the GPS because in many cases they would already know how to arrive to the supermarket. In spite of this, most users considered that the context-based adaptation was useful and the interaction adaptation for routine tasks was adequate. Furthermore, the users considered the system to be good for long-term objectives because it could make their lives more comfortable and help them to optimize natural resources. In addition, most of them commented that they would like to have this system to automate their home routine tasks.

Although the initial results of the experiment show that by following our approach we can greatly help users carry out their routines, additional experimentation would be required to analyse the user acceptance of the system for longer periods. Due to time and resource constraints, we used a device simulator to provoke the opportune context changes and gave the users the list of the steps that they should perform to reproduce the specific tasks and contexts of use. However, we plan to evaluate the system in a more realistic context where tasks are competing with daily activities.

Finally, we conclude that the approach achieves satisfactory results regarding unobtrusive automation of routines. Taking into account the obtrusiveness of each task, we enhance the user experience of routine automation. Nevertheless, we must provide users with more control over adaptations; otherwise, users will not be comfortable with them even though they achieve a high level of consideration. Allowing the user to configure or personalize the adaptations to different situations would be a very useful feature to fit all user needs and preferences.

8. Discussion

The presented method provides abstract models that allow the requirements related to the routines and the desired level of obtrusiveness 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 [41]. Thus, designers can focus on the main concepts (the abstractions, such as User, Task, Location, etc.) without being confused by many low-level details [41]. This makes the provided models easier to specify, understand, and maintain than source code.

In addition, the developed software infrastructure directly interprets the models at runtime in order to automate the routines in the opportune context situation. Thus, the models become the primary means to understand, interact with, and modify the routines 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 software infrastructure [42].

However, as the evaluation results show, the proposed method also has some drawbacks that should be improved in further work. First, we plan to improve the task model modelling tool in order to allow the loading of the context ontology model to show the possible context properties to use in the context conditions. This improvement aims to prevent the errors in the names of the corresponding context ontology properties used. Second, although the execution of the routines takes into account the user preferences, they may change over time and the user should be able to evolve and personalize both the routines and their obtrusiveness level. An end-user graphical tool like the one presented in [43] can be used for this purpose. This tool allows end-users to change the user routines by updating the models at runtime. It can be extended so that users can configure what is already designed and also add personal unobtrusive automations that could not have been taken into account before. Thus, the system receives feedback to adapt according to user requirements. It is worth noting that, since our approach allows evolution to be performed by simply changing the models, this tool will not have to change any line of code to apply user personalization. In addition, since obtrusiveness is handled as a separate concern, a given pervasive service can be presented to

the user in a completely different manner by only changing the obtrusiveness specification, without altering the routine description.

To provide more automation in this evolution process, the software infrastructure could also provide self-learning capabilities. Machine-learning algorithms could automatically detect changes in user behaviour that require evolving the models [44]. For instance, the infrastructure could detect if users execute tasks to counteract any automated task (e.g., after the system raises the blinds when the user enters in the room, the user lowers the blinds because s/he prefers the lights of the room to be switched on). The counteracted tasks cannot be automated any more. To achieve this, the services provided by the pervasive system should provide information about which operations perform contradictory tasks.

Finally, the presented work could use the Model-based Reconfiguration Engine proposed in [45] to handle possible failures and self-heal accordingly. Designers can provide reconfiguration rules to indicate the behaviour of the system when a failure is detected. When this happens, the engine executes the corresponding rules for reconfiguring the system. For instance, if a service that must be executed in a routine is not working, the engine could be used to reconfigure the system to find another service that can perform the same task.

9. Conclusions

In this work, we have presented and evaluated a method for the automation of routine tasks in an unobtrusive manner. This method adjusts the obtrusiveness level of each executed task at runtime in such a way that it demands the appropriate user attention when it is actually required. To achieve this, the method provides abstract models that allow the requirements related to the routines and the desired levels of obtrusiveness to be captured. These models are also used at runtime. We have provided a software infrastructure that directly interprets them in order to automate the routines in the opportune context situation.

Further work will be dedicated to extending the end-user tool presented in [43] in order to enable users to evolve and personalize both the routines and their obtrusiveness level. Also, we will extend the software infrastructure to provide self-healing capability to prevent failures and to provide self-learning capability of interaction obtrusiveness based on user behaviour.

To conclude, we believe that the use of models can provide many benefits to the heterogeneous and rapidly changing pervasive systems field. The method presented in this work to achieve the automation of unobtrusive routine tasks is our contribution to this vision.

Acknowledgment

This work has been developed with the support of MICINN under the project EVERYWARE TIN2010-18011 and co-financed with ERDF, in the grants program FPU, and it has also been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria.

References

[1] F. Mattern, The vision and technical foundations of ubiquitous computing, Upgrade 2 (5) (2001) 2–6.
[2] D. T. Neal, W. Wood, Automaticity in situ and in the lab: The nature of habit in daily life, in: Oxford Handbook of Human Action, Oxford University Press, 2009.
[3] M. Tedre, What should be automated?, interactions 15 (5) (2008) 47–49.
[4] W. W. Gibbs, Considerate computing, Scientific American 292 (1) (2005) 54–61.
[5] D. Garlan, D. Siewiorek, A. Smailagic, P. Steenkiste, Project aura: Toward distraction-free pervasive computing, IEEE Pervasive Computing 1 (2002) 22–31. doi:http://doi.ieeecomputersociety.org/10.1109/MPRV.2002.1012334.
[6] D. Cook, M. Youngblood, I. Heierman, E.O., K. Gopalratnam, S. Rao, A. Litvin, F. Khawaja, Mavhome: an agent-based smart home, in: Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on, 2003, pp. 521 – 524. doi:10.1109/PERCOM.2003.1192783.
[7] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, H. Duman, Creating an ambient-intelligence environment using embedded agents, IEEE Intelligent Systems 19(6) (2004) 12–20.

[8] P. Rashidi, D. J. Cook, Keeping the resident in the loop: adapting the smart home to the user, Trans. Sys. Man Cyber. Part A 39 (5) (2009) 949–959. doi:10.1109/TSMCA.2009.2025137.
URL http://dx.doi.org/10.1109/TSMCA.2009.2025137

[9] K. Henricksen, J. Indulska, A. Rakotonirainy, Using context and preferences to implement self-adapting pervasive computing applications: Experiences with auto-adaptive and reconfigurable systems, Softw. Pract. Exper. 36 (11-12) (2006) 1307–1330. doi:10.1002/spe.v36:11/12.
URL http://dx.doi.org/10.1002/spe.v36:11/12

[10] M. García-Herranz, P. A. Haya, X. Alamán, Towards a ubiquitous end-user programming system for smart spaces, Journal of Universal Computer Science (JUCS) 16 (12) (2010) 1633–1649.

[11] A. Aztiria, A. Izaguirre, J. C. Augusto, Learning patterns in ambient intelligence environments: a survey, Artif. Intell. Rev. 34 (1) (2010) 35–51. doi:10.1007/s10462-010-9160-3.
URL http://dx.doi.org/10.1007/s10462-010-9160-3

[12] F. Castanedo, J. García, M. A. Patricio, J. M. Molina, A multi-agent architecture based on the bdi model for data fusion in visual sensor networks, Journal of Intelligent and Robotic Systems 62 (3-4) (2011) 299–328.

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

[14] R. Hervás, J. Bravo, J. Fontecha, Awareness marks: adaptive services through user interactions with augmented objects, Personal Ubiquitous Computing 15 (2011) 409–418.

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

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

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

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

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

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

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

[22] E. Serral, L. Sabatucci, C. Leonardi, P. Valderas, A. Susi, M. Zancanaro, V. Pelechano, Incorporating users into ami system design: from requirements towards automation, in: Proceedings of the 20th International Conference on Information Systems Development (ISD 2011), 2011.

[23] P. Johnson, Tasks and situations: considerations for models and design principles in human computer interaction, in: HCI International, 1999, pp. 1199–1204.

[24] F. Paternò, Concurtasktrees: An engineered approach to model-based design of interactive systems, in: L. E. Associates (Ed.), The Handbook of Analysis for Human-Computer Interaction, 2002, pp. 483–500.

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

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

[27] W. Buxton, Integrating the periphery and context: A new taxonomy of telematics, in: Proceedings of the Graphics Interface Conference, Morgan Kaufman, 1995, pp. 239–246.

[28] M. D. D. Fabro, J. Bzivin, P. Valduriez, Weaving models with the eclipse amw plugin, Eclipse Modeling Symposium.

[29] B. Kitchenham, L. Pickard, S. L. Pfleeger, Case studies for method and tool evaluation, IEEE Softw. 12 (4) (1995) 52–62.

[30] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering: an introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[31] Goal/question/measures, http://www.gqm.nl/.

[32] C. Jones, Applied Software Measurement: Assuring Productivity and Quality, McGraw-Hill, 1991.

[33] Web of the experiment, http://www.pros.upv.es/adaptio/methodevaluation.

[34] Statistical analyses using spss, http://www.ats.ucla.edu/stat/spss/whatstat/whatstat.htm#1sampt.

[35] T. Strang, C. Linnhoff-Popien, 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, 2004.

[36] J. De Bruijn, Using ontologies - enabling knowledge sharing and reuse on the semantic web, Technical report, Digital Enterprise Research Institute (DERI) (Oct. 2003).

[37] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, A. Solberg, Models@ run.time to support dynamic adaptation, Computer 42 (10) (2009) 44 –51.

[38] G. K. Mostéfaoui, J. Pasquier-Rocha, P. Brézillon, Context-aware computing: A guide for the pervasive computing community, in: Proceedings of the IEEE/ACS International Conference on Pervasive Services (ICPS04), 2004, pp. 39–48.

[39] P. Runeson, M. Hst, Guidelines for conducting and reporting case study research in software engineering, Empirical Soft. Eng. 14 (2) (2009) 131–164.
[40] J. R. Lewis, Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use, Int. J. Hum.-Comput. Interact. 7 (1) (1995) 57–78.
[41] F. Paternò, From model-based to natural development, HCI International (2003) 592–596.
[42] E. Serral, P. Valderas, V. Pelechano, Supporting runtime system evolution to adapt to user behaviour, in: Proceedings of the 22nd international conference on Advanced information systems engineering, CAiSE'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 378–392.
URL http://dl.acm.org/citation.cfm?id=1883784.1883823
[43] E. Serral, F. Pérez, P. Valderas, V. Pelechano, An end-user tool for adapting home automation to user behaviour at runtime, UCAmI'10 (2010) 201–210.
[44] E. Serral, P. Valderas, V. Pelechano, Improving the cold-start problem in user task automation by using models at runtime, in: Information Systems Development, 2011, pp. 671–683.
[45] C. Cetina, P. Giner, J. Fons, V. Pelechano, Autonomic computing through reuse of variability models at runtime: The case of smart homes, Computer 42 (2009) 37–43.