Making Task Models and Dialog Graphs Suitable for Generating Assistive and Adaptable User Interfaces for Smart Environments

Michael Zaki and Peter Forbrig
Department of Computer Science, Rostock University, Albert-Einstein Str.22, Rostock, Germany

Keywords: Smart Meeting Room, Explicit Interaction, Implicit Interaction, Assistive User Interface, Task Model, Dialog Graph, Activity Pattern, HCI Pattern, User Profile.

Abstract: Nowadays smart environments are gaining special attention among the various ubiquitous computing environment types. The main tenet of a given smart environment is to deliver proper assistance to the resident users while performing their daily life tasks. The environment aims to implicitly infer the user's intention and based on that information, it offers the optimal feasible assistance which helps the user performing his/her task. Task models seem to be a convenient starting point for developing applications for those environments, as they give the developer the opportunity to focus on the user tasks to be assisted. Already existing approaches offer solutions to make the transition between task models and the final user interfaces. However, smart environments are dynamic environments in which the inclusion of new user or device types is probable. Consequently, an optimal application to be operated in such an environment is required to consider the extensibility aspect within its design. Additionally, the implicit interaction technique has to be taken into account. Thus, in this paper we provide an attempt to include the implicit interaction paradigm within the design of our application as well as to ensure the extensibility needed to encounter the variation of the surrounding environmental settings.

1 INTRODUCTION

In the last few decades, the topic of smart environments has become a hot research field. A lot of work has been accomplished by researchers in order to develop prototypes of such environments assisting the users performing their daily life tasks (e.g. (Das et al., 2002), (Bobick et al., 1999), (Srivastava et al., 2001) and (Waibel et al., 2003)). The notion of smart environment has been inspired by the work of (Weiser, 1999), when he motivated the idea of ubiquitous computing environments by declaring that *"Machines that fit the human environment instead of forcing humans to enter theirs will make using a computer as refreshing as taking a walk in the woods"*. Afterwards, (Cook and Das, 2004) defined the term "smart environment" as a *"a small world where different kinds of smart devices are continuously working to make inhabitants' lives more comfortable"*. In contrast to various approaches in the HCI field adopting an explicit user-machine interaction technique, several attempts suggesting an implicit interaction technique between the user and the hidden application have also been presented (Schmidt, 2000). Implicit interaction as defined by Schmidt is *"an action, performed by the user that is not primarily aimed to interact with a computerized system but which such a system understands as input"*. The main advantage offered by the implicit interaction paradigm is the application's transparency. The resident users do not have to do extra-tasks in order to ask for assistance (e.g. pressing a button on a user interface). Instead, the environment is equipped with sensors perceiving the user and the state of the environment, and based on that information the system can infer the task being currently executed by the user and thus acts accordingly. According to (Kirste et al., 2001) the level of smartness associated to a given environment is measured by its capability to react to the user's objectives and not to pure sensor data.

Despite the many advantages of the implicit interaction technique, experiments have shown that users usually prefer to have control over the system they are using. The user gets a negative impression about the application if he/she feels being controlled by the environment. Moreover, if the system fails to infer the right intention of the user, then the application cannot react in the expected way which can be very distracting for the user. Therefore, getting continuous feedback from the end user and giving him/her the op-

Zaki M. and Forbrig P..
Making Task Models and Dialog Graphs Suitable for Generating Assistive and Adaptable User Interfaces for Smart Environments.
DOI: 10.5220/0004315100660075
In *Proceedings of the 3rd International Conference on Pervasive Embedded Computing and Communication Systems* (PECCS-2013), pages 66-75
ISBN: 978-989-8565-43-3
Copyright © 2013 SCITEPRESS (Science and Technology Publications, Lda.)

portunity to explicitly adjust the system is still highly desirable. From our point of view, an optimal design of the application to be operated in a given smart environment has to satisfy the required equilibration between the explicit and implicit interaction paradigms. Such an equilibration enables the minimization of the burden of performing the tasks without totally taking the control from the user.

A pre-requisite for developing an application for smart environments is to analyze the nature of tasks to be performed by the resident users. Only a thorough understanding of the activities the user wants to perform and the goal he/she wants to achieve enables the application designer to provide the proper assistance to be offered for that user. Therefore, task models seem to be a convenient starting point for the development of such an application, since they give the developer the opportunity to focus on the user's tasks. Whereas task models have initially been introduced as a tool to elicit requirements in the early analysis stages (Kato and Ogawa, 1993), they are nowadays considered as an appropriate starting point for interactive processes development (Caffiau et al., 2007; Paterno, 2000). Various task model notations exist (e.g. HTA (Annett and Duncan, 1967), GOMS (Card et al., 2000), CTT (Paternò et al., 1997),UAN (Hartson and Gray, 1992) ,...etc.). Considering its hierarchical structure and its huge set of temporal operators, CTT has become one of the most widely used task model notations as it gives the developer the opportunity to define the execution order of the tasks to be performed.

In the context of our work, we strive to develop an application to be operated in a smart meeting room (smart lab in our university). Smart meeting rooms are a specific subset of smart environments where the main goal is the beneficial exchange of information among the resident users (Nijholt et al., 2004). Several requirements have to be fulfilled by the desired application in order to achieve the optimal conceivable assistive system which then leads to the user's satisfaction with the environment. Among those requirements, we focus in this paper on two crucial points: The application's adaptability and scalability and also the equilibration between the explicit and the implicit interaction techniques and the smooth transition between them. As already discussed, a typical smart environment is enriched with various devices and user types which are supposed to cooperate together in order to achieve a final shared team goal. The application should be capable of providing the needed assistance for every individual within the environment. However, the individuals are not identical and can be categorized according to their personal characteristics (modality preference, impairment type (if any), level of expertise,...etc). Moreover, the role being played by every user determines the range of possible system interventions. For instance, if we take the example of a smart meeting room and consider the simple scenario of having a presentation, it is noticeable that the assistance to be offered by the room to a specific user at a given time 't' depends on this user's profile (i.e., personal characteristics), the task he/she is performing at that time and the platform or device on which the interface is to be rendered. It is noteworthy to mention that our ultimate goal is to generate individualized user interfaces for every actor in the room. In other words, the view displayed for every user to help him/her performing a given task has to be compatible with this user's personal preferences and desires and has also to change whenever he/she starts to perform another task.

Consequently, the application's adaptability to the different user and device types is of great interest if we aim to achieve a tailored assistance to be offered for every user within the environment depending on his/her profile. In the context of our work, we do not refer to "adaptability" only as the ability of the user interface to get adapted to the various predefined user and device categories, but also the application's extensibility and its ability to get adjusted according to emerging factors which were unforseen at the primary design phase. We aim to minimize the effort needed by the designer if later on he/she decided to extend the application to serve a totally new user category or device type.

In this paper, we briefly discuss the flow of the development of our application for smart meeting rooms and we propose useful extensions to task models and dialog graphs (Schlungbaum and Elwert, 1996) in order to make it feasible to achieve an adaptable and extensible assistive system where a smooth merging between explicit and implicit interaction techniques is realized. The paper is structured as follows: In the next section, previous related approaches and relevant work are being discussed. Section 3 starts by highlighthing the needed features in task models and dialog graphs and motivates the extensions which are then thoroughly explained in subsections 3.1 and 3.2. In those subsections, the application's adaptability and the notion of hybrid interaction technique are tackled in more details. Afterwards, in section 4 an illustrative example is presented to assimilate the functionality of the assistive application, and also to clarify the contribution of the paper and the benefit gained out of the suggested extensions. Finally, we conclude the paper and we summarize the main points we discussed and the contribution of our work.

2 RELATED WORK

As already mentioned, the usage of task models has evolved from being just a tool to extract the needed software requirements to a useful basis on which the development of interactive user interfaces can be founded. In the literature, several attempts to derive final user interfaces out of task models exist (Limbourg and Vanderdonckt, 2003; Wolff and Forbrig, 2009). In general, those attempts follow a model-driven development approach. In (Sousa et al., 2010) the business alignment framework is introduced where the development process starts by a task model and a supplementary domain model. Afterwards an abstract user interface (AUI) is generated which then should be employed to derive concrete UI components.

In the context of smart environments, the interaction can be divided in explicit or implicit interaction (Sara, 2009). Whereas the above mentioned approaches are aiming to a final user interface where the tasks should be explicitly triggered by the end user (explicit interaction), another feasible interaction technique can be realized by reasoning about the contextual data from the surrouding sensors (implicit interaction). According to (Ju and Leifer, 2008) , even if it is possible to interact using only one of those techniques, both paradigms are needed to implement a robust and usable system. In Section 3, it is described in further details how we suggest to design our application based on a conceivable merging process between both interaction paradigms.

The term task can be described as an action the user performs in order to achieve a desired goal. Task models in general aim to represent the tasks to be carried out by the user. Being the most used notation for task models, CTT was prone to several improving attempts. Initially, CTT distinguished five task types namely “abstract”, “user”, “interaction”,‘application” and “cooperation” tasks. However, several extensions took place in order to make the notation more suitable for specific purposes. For example, the authors in (Klug and Kangasharju, 2005) extended the ConcurTaskTree (CTT) notation to allow the dynamic execution of task models. Moreover, in (Bergh and Coninx, 2004) an approach extending the CTT notation in order to integrate the dynamic context within the task-based design is proposed. As it will be shown in subsection 3.1, we also extend the already existing CTT task types to make the automation of the adaptation process feasible for our final application.

Usually, after the resulting task model of the analysis phase is modeled, further iterative transformation processes take place until we reach the most fine-grained task model in the lowest level of the design stage where all the technical details of the application are defined (Wurdel et al., 2008b). According to (Wolff et al., 2005), the resulting model can be employed in addition to other environmental models in order to derive a dialog graph. The dialog model can afterwards be transformed to an abstract user interface to be tailored based on the encountered situation and finally results in the desired concrete user interface. Fig. 1 illustrates the meta-model of a dialog graph as it has been introduced by (Wolff and Forbrig, 2009). As it is shown, every dialog graph is composed of a group of dialog views and transitions between those views. Only two transition types already exist, namely the sequential and the concurrent transitions. Additionally, input and output ports are attached to the different dialog views. Wheareas a dialog view’s input port gets the transition from the previous dialog view, the output port is responsible to link the current dialog view to the next one to be displayed. According to (Luyten et al., 2003) , a dialog model is defining a sequence of user interactions, or activity chain that has to be followed to achieve the desired goal. However, within the existing dialog graphs approaches, the transition between the views can only be triggered explicitly when the end user performs a specific event. In the context of smart environment applications, there is an emerging need for the inclusion of the implicit transition possibility in the dialog models. This aspect is tackled in more details in subsection 3.2.

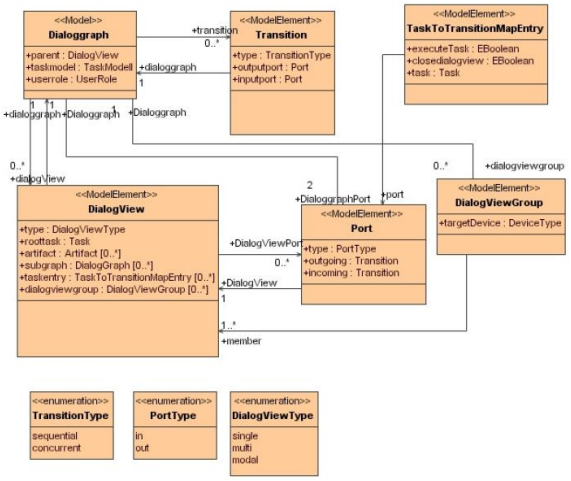

Figure 1: Meta-model of a dialog graph as presented by Wolff.

3 PRACTICAL EXTENSIONS FOR TASK MODELS AND DIALOG GRAPHS

First of all, we start by a brief explanation of the interaction technique our application aims to realize in the room. We strive to provide an optimal hybrid interaction technique minimizing the burden on the end user but still providing him/her the control over the system. Therefore, our idea is to implicitly infer the role to be played by every individual in the room, and depending on his/her role the corresponding assistance is provided through his/her own user interface. This approach is better than having a fixed user interface with all assistance options provided by the room, since our user interface has the ability to adapt itself according to the task the user is currently performing and consequently the user is not bothered with a crowded user interface where a lot of enabled tasks are not relevant to the goal he/she wants to achieve. In other words, the room acts neither in a complete implicit way by providing the assistance required, nor following a total explicit interaction paradigm where the user has to specify everything. Instead, the room implicitly infers the goal the user wants to achieve, and displays only the relevant supporting tasks from which the user can take benefit.

As already discussed, the design of a supportive system for smart environments has to focus on the tasks the user wants to perform and to follow the user-centered design guidelines. Therefore, we started our system's development process which is captured in Fig. 2, by analyzing the user's behavior in our smart meeting room and collecting information about the various tasks he/she may want to perform in the context of various scenarios which may take place in such a room (e.g. conference session, parallel presentation session, work defense, video session,...etc.). For each one of those scenarios, various roles have to be played by the resident actors in order to successfully achieve the final desired common goal. According to (Wurdel et al., 2008a), approximating the user's behavior inside the room to the role he/she plays is an acceptable assumption. Thus, we compiled the extracted roles in the form of task models. Afterwards, we iteratively convert this model to a design level one by integrating step by step the assistance the room can provide for every task to be carried out by the user. Once we achieve the design level model, a transformation to dialog graphs describing the logical sequence of transitions between the different views can take place. Finally, a simple mapping process of the resulting dialog model to final UI components is feasible. An important point to mention is that the transformation from task models to dialog graphs used to be a task for the designer as he/she has to decide which tasks should be included within the same views and how exactly the transition between all of those views have to occur. However, as our ultimate goal is to automate the conversion from the initially compiled task models to the final user interfaces, we follow some heuristics that are presented in subsection 3.2 in order to bridge the gap between the task model in the design level and the corresponding dialog graph.

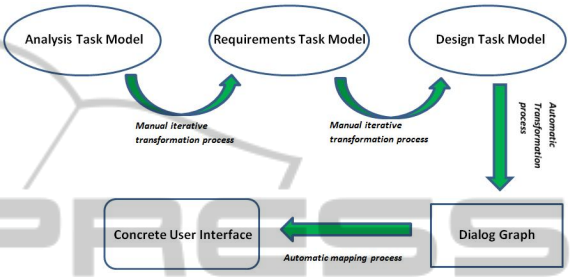

Figure 2: Our application's development flow.

Smart environments are dynamic environments in which the entrance of new device types is probable. Moreover, the user characteristics have a direct influence on the way the assistance offered by the room is about to be displayed to the user. Even if we take the current conceivable user characteristics into account while designing our application, new relevant characteristics may be discovered in the future and thus the application will have to be extended in order to encompass those new characteristics as well. Therefore, an optimal solution from our point of view is to formalize the changes that occur to the tasks influenced by a given user characteristic in the form of task patterns abstracting from the specific context of use. More details and examples of such a formalization are to be discussed in the next subsection. In fact, the notion of task patterns is not totally new. After the term "pattern" was initially introduced by (Alexander, 1977) in the domain of urban architecture, patterns have successfully made their way to the software engineering (Gamma et al., 1995) as well as the HCI field (Borchers, 2001). In their simplest form, task patterns have been known as "task templates" when they were first presented by Breedvelt in (Breedvelt-Schouten and Severijns, 1997). Afterwards, several approaches extending this notion have been suggested e.g. (Paterno, 2001; Wurdel et al., 2007; Radeke, 2007; Zaki et al., 2011).

In a nutshell, in subsection 3.1, we tackle the application's adaptability issue by suggesting some useful extensions to the CTT notation and presenting our idea of defining different user characteristics's effects

on the tasks performance in the format of task patterns. On the other hand, in subsection 3.2, we further elaborate the extensions we suggest to the dialog graph notation to enable the integration of the implicit interaction paradigm in our development flow of the application.

3.1 Making Task Models Adaptable to User Characteristics

No matter how comprehensive the application we design is in terms of the user characteristics considered, the environment is always subtle to the inclusion of a new relevant characteristic or user group type. Therefore, the application's extensibility is a crucial factor to be taken into account. Since the developer can only consider the already known influential user characteristics while developing the application, we need a methodology enabling the inclusion of any given new aspect which was unforeseen at design time. The main motivation is to minimize the work and effort the developer will have to make in order to adjust the system with respect to the newly introduced aspect.

In our work, we noticed that every specific user characteristic (regardeless its type) is affecting the task model in a systematic way. In other words, any given characteristic influences specific task types (referring to the CTT notation) and changes them in a repetitive manner, while other task types remain unchangeable. To make the point more clear, let us consider the user impairment type criterium. It is noteworthy to mention that despite the low probability of having impaired users in a smart meeting room, the application we develop in our lab should then be generalized and is supposed to work in other smart environment types as well. Smart homes for example are environments in which the probability of existence of elderly or impaired people is relatively high. In Fig. 3, a part of the task model at the design level representing the tasks to be carried out by a default listener attending a presentation session is depicted. From the figure, one can notice that the listener has to sit and listen to the talk, take his/her own notes and then the system is going to offer the opportunity to view the slides of the presentation or even related information on the user's own user interface.

How would the model in Fig. 3 look like in case the user we want to support is suffering from some hearing problems? In our earlier work in (Zaki and Forbrig, 2011), we introduced the so-called "user-oriented accessibility patterns" where some task templates are suggested and which are supposed to be manually used by the designer of the application in case he/she prefers to take impaired people into account within the design. Briefly, (Zaki and Forbrig, 2011) suggests changing the modality of the information from the initial one that the user cannot process to a new one which can be handled by the impaired user. This modality conversion is represented using a task template which can be reused whenever we want to model the case of a user suffering from the same problem. Thus, it is possible to represent the user impairment type criterium in the form of one or many task templates. As an example, the task template to be employed for replacing a task of type "user" in case the user is receiving input (information) from the environment is depicted in Fig. 4.

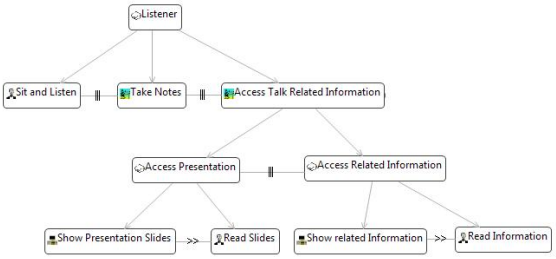

Figure 3: Part of the listener task model in the design level.

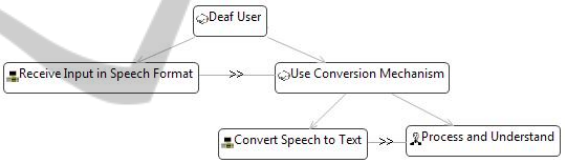

Figure 4: user-input accessibility pattern.

Moreover, by applying this pattern on the task model example in Fig. 3, we get the resulting model in Fig. 5. It is noticeable that only the task "sit and listen" has been changed while the other tasks remain the same. This is due to the fact that a deaf user is able to see the slides, take notes and even browse the information provided by the assistive user interface without any problems. However, he/she cannot listen to the presentation and therefore a conversion mechanism is needed to change the modality of the information to text (using a software). To summarize, whenever the developer wants to add a new user characteristic to be considered, he/she will have to determine which task types (e.g. user, interaction, application,...etc.) are affected by the changes and build a generic task model (task template) to replace those tasks.

As already mentioned, the main goal of defining a corresponding task template to be employed for every newly introduced user characteristic is to minimize the time and the effort spent by the designer to integrate a new characteristic. However, to achieve

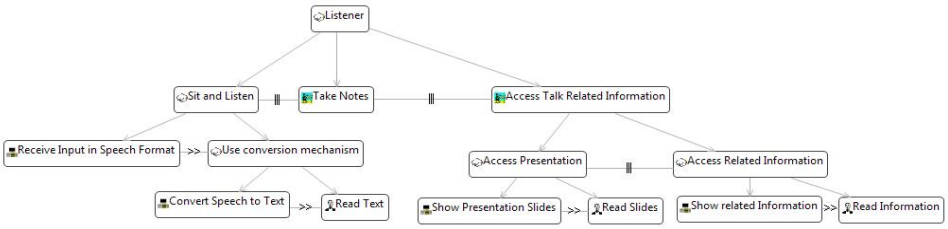

Figure 5: Part of the listener task model in case of a deaf user.

this goal an automation of the transformation process from task models to final user interfaces is mandatory. Such automation also guarantees that whenever the application has to be adjusted to cope with a new discovered task template or even user characteristic, the designer will have to make changes only on the task model level and then the application will be able to adapt itself accordingly. To achieve the desired automation, more information about the nature of tasks is required. (Paternò et al., 1997) categorizes the tasks depending on the allocation of their performance. A pre-requisite for our work is to have more information about the tasks included within the task model which is about to get transformed. The information we mainly need is related to the effect of this task on the environment. "Walking from door zone to presentation zone", "give talk" and "listen to talk" are three tasks of the same type "user". Nevertheless, if we want to automate the updating process of the existing task models according to the user impairment type for example, we must be able to differentiate between those tasks. The reason is that when the user walks in the room, only his/her location is changing but no exchange of information with surrounding environment is taking place. On the other hand, the user giving a talk is providing information to the listeners which are resident entities in the environment. Also the listeners are getting information from another entity (the presenter) in the environment. Consequently, if for example the user has some hearing problems, only the task "listen to talk" has to be replaced by the new template. As a result, we suggest to subcategorize the "user" task type to three distinguishable subcategories namely the user task (the one existing in CTT), the user output task and the user input task. Similarly, we differentiate between the so-called "display application task" which is an application task displaying output to the user, and the "computational application task" which is for internal processing and computing. In Fig. 6, the new task types are presented. We built task models for the user's activity patterns we extracted in the analysis phase and we used the new task types since that stage. In that way, we are able to adapt our task models to any new incident or criterium by just defining through a wizard the task types that are affected by the changes and the corresponding template to replace those tasks. Afterwards, the user interface is automatically adapted to consider the desired case. An example of a task model where the new suggested task types are employed is presented in the context of the system's application example discussed in section 4.

Type	Symbol	Description
User Output task		The user is providing output (information) to other users in the environment, without interacting with the system.
User Input Task		The user is receiving input (information) from other users in the environment, without interacting with the system.
Display Application Task		This task is performed by the system, after receiving some internal information. This task results in an output to be displayed to the user.
Computational Application Task		This task symbolizes an internal computation performed by the system without providing any output to the user.

Figure 6: Introduced CTT task types.

3.2 Making Dialog Graphs Suitable for Implicit Interaction

As already discussed, dialog graphs (Schlungbaum and Elwert, 1996) are an efficient way to illustrate the logical flow of the application in an abstract way. They give the developer an overview of the different views which are about to be displayed to the user and the way the transition between those views is going to take place. Thus, the developer is provided with a comprehensive idea about the user-application interaction means.

For the design of our application, such a graph is of great interest. However, those dialog models have been usually employed in the context of interactive applications where the transition among the views is supposed to occur due to an explicit intervention by the user (e.g. pressing a button, asking for a service,...etc.). Therefore, only two transition types exist namely the "sequential transition" referring to a transition from one view to another, where the initial view disappears, and the "concurrent transition" indicating a transition from one view to another where the initial view remains visible but inactive.

Since we want to make it feasible to automatically change the views when the room infers that a

given environmental condition is met (e.g. the user changes his position), new transition types enabling such an implicit migration between the views is required. Consequently, in addition to the already existing explicit transitions, namely "sequential transition" and "concurrent transition", we introduce two new transition types namely the "implicit sequential transition" and the "implicit concurrent transition" which are depicted in Fig. 7.

Type	Symbol	Description
Implicit Sequential Transition		Identifies a transition implicitly triggered by an environmental event. The target view becomes visible while the source view disappears.
Implicit Concurrent Transition		Identifies a transition implicitly triggered by an environmental event. The target view becomes visible while the source view remains visible but inactive.

Figure 7: Implicit dialog graph transitions.

From the above figure, it is noticed that the "implicit sequential transition" is identified by the empty ellipse and coloured triangle, while the "implicit concurrent transition" is symbolized by an empty ellipse and empty triangle. An example of a dialog graph where the new suggested transition types are employed is presented in the context of the system's application example discussed in the next section.

Another pre-requisite for automating the whole process is to automate the conversion of task models to dialog graphs. Nevertheless, this conversion has to be guided by well-defined rules to guarantee an optimal distribution of the tasks (extracted from the task model) into the various dialog views constructing the resulting dialog graph. For that purpose, we take advantage of the heuristics provided by (Paterno and Santoro, 2002) as well as new suggested guidelines in order to seamlessly derive the desired dialog graphs out of the existing task models.

In a nutshell, we start by collecting the so-called enabled task sets (ETS). An ETS is defined in (Paterno, 2000) to be *" a set of tasks that are logically enabled to start their performance during the same period of time"*.

Afterwards, heuristics are followed in order to minimize the number of ETSs gathered. We accomplish that by merging some of the resulting ETSs which are semantically related and thus can be presented together within the same dialog view. This merging process forms sets of ETSs, namely Task Sets (Paterno and Santoro, 2002). In the following, the guidelines and heuristics controlling the conversion process to the desired dialog graphs are provided (From 1 to 4 are the heuristics adopted from (Paterno and Santoro, 2002), while our new suggested heuristics are from 5 to 9):

1. If two ETSs differ for only one element, and those elements are at the same level connected with an enabling operator, they could be joined together with the ordering operator.
2. If an ETS is composed of just one element, it should be joined with another ETS that shares some semantic feature.
3. If some ETSs share most elements, they could be unified.
4. If there is an exchange of information between two tasks, they can be put in the same ETS in order to highlight such data transfer.
5. The "user tasks" are not taken into account on the views, as no visualization of such tasks is required.
6. Concerning the application tasks, the "display application task" results in an object to be displayed, and thus it should be considered in the view. However, a "computational application task" is ignored and removed from the ETS.
7. In order to keep a view even after making a transition to a subsequent one, we have to use a concurrent transition and an "exit" task should be added to the initial view in order to ensure a possible exit out of that view.
8. If we have one of the enabled sets having only one "user task", then this means that this enable set results in an implicit transition between the previous view and the next one and the transition takes place at runtime whenever the post-condition of the user task is realized.
9. If within the ETS there are redundant tasks, then we represent the parent tasks within one view so that the user chooses between the parent tasks and get redirected to new views were no redundant tasks exist.

4 APPLICATION EXAMPLE

In this section, we provide a very brief walkthrough of our application where we highlight the contribution of the extensions we made for task models and dialog graphs. Let us assume we have a conference session to be held in a smart meeting room. The conference session case is one of the main cases we covered. We extracted for that case the corresponding task patterns for all roles which may take place. For the sake of brevity, we only focus here on the role "presenter".

After collecting the tasks a presenter has to perform in the context of a conference session, we iteratively refine this model until we reach the design level. Due to the lack of space, only a part of the resulting

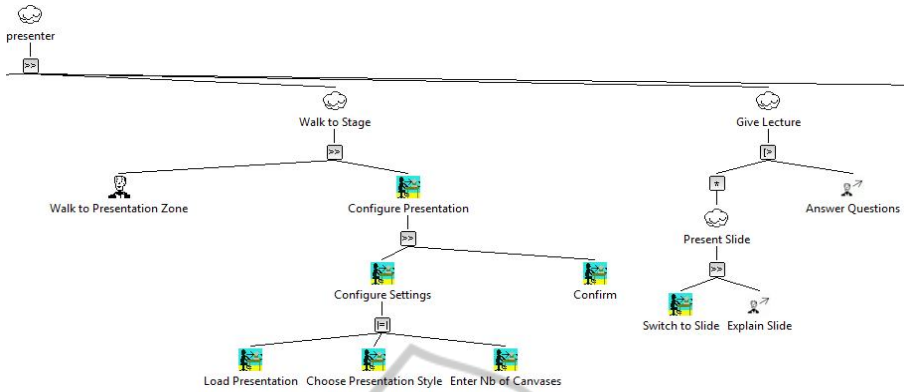

Figure 8: A part of the presenter task model in the design level.

model in the design level is illustrated in Fig.8. We employ the editor developed by (Wolff et al., 2005) to create the task models and dialog graphs presented within the example. The figure above depicts only two of the tasks which are carried out by a presenter. If the user wants to give a talk, he/she will walk to the presentation zone and then start to give the talk. In the figure above, it is distinguished between the user task "Walk to Presentation Zone" and the user output tasks "explain slide" and "answer questions". If the developer then wants to take deaf (also not speaking) users into account, then he/she has just to define the task template (accessibility pattern) to be employed and has to specify that only tasks of type "user output" should be subject to those changes.

In that way, the task model will be adapted in a successful way. Thus, the developer takes advantage of the user task types categorization. Now, we investigate in more details the abstract task "Walk to Stage". When the system infers through the ambient sensors that the task "Walk to Presentation Zone" has already been executed (presenter is now at presentation zone), the assistive user interface should then start to display the presentation configuration settings as it is shown in Fig.8. The presenter has thus the opportunity to choose the number of projectors to be used and also the preferred presentation style to present his/her slides within the room.

Now let us assume we want to move to the next step by converting the task model we have to a corresponding dialog graph as a transitional step for deriving the final required user interface. The tasks within the presenter's task model should then be distributed over the various views following the heuristics defined in subsection 3.2. Fig.9 depicts a part of the dialog graph for the role presenter. If we take a look at the "Connection" view as an example, it contains the tasks "Register", "Identify Himself" and "Connect as Default User" which are logically related as they all aim to specify the user's profile in order to adapt the assistance according to it. The "Register" task has an output port leading to the "Sign Up" view which means that the user has to create his/her user profile, while the "Identify Himself" task has an output port leading to the "Sign In" view in order to load the existing user's profile in case he/she is already registered.

Figure 9: Part of the dialog graph for the role presenter.

In Fig.9, one can additionally notice the implicit sequential transition between the connection view and the presentation styles view. This transition indicates that the "presentation styles" view is not explicitly achieved when the user presses a given button on the "connection" view. Instead, the transition is taking place after a specific environmental precondition is satisfied (presenter is at presentation zone in this case). Thus, having those newly introduced implicit transitions enables the developer to express an implicitly triggered transition which in the context of our application is of high interest.

To make the idea of the assistive system more concrete to the reader, Fig.10 illustrates the supportive user interface corresponding to the "Presentation

The window depicted in Fig.10 gives the presenter the opportunity to upload his/her slides to be presented, to choose the number of canvases to be employed for the presentation and also the style in which the slides should be visualized within the room. The presenter can choose between the default presentation

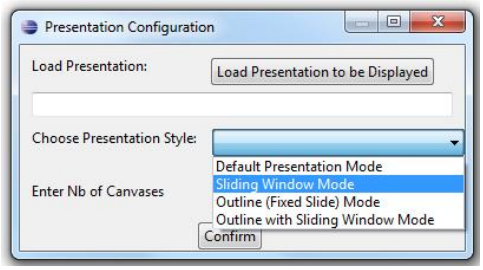

Figure 10: A screenshot of the resulting supportive user interface.

mode, another mode where the outline is always displayed on one of the existing screens, the sliding window mode where the slides are turning in a circle between the various canvases or even a combination between two of the existing modes.

5 CONCLUSIONS

In this paper, we presented an attempt for extending the CTT task model notation as well as dialog graphs in order to make them suitable for the design of smart environment applications. We started by giving an overview of the domain of smart environments. Then, we motivated our choice of a hybrid interaction technique to be adopted by the application operated in such environments. We clarified that having an equilibration between the explicit and implicit interaction paradigms leads to an optimal assistance provided by the room, where the user can take benefit of the supporting ability of the room without losing his/her control over it. Additionally, we justified the choice of task models as a starting point for the development of the desired application and we presented the development flow we follow in order to achieve the final user interfaces out of the task models we collected in the analysis phase. Afterwards, we tackled another crucial requirement to be satisfied, which is the application's adaptability and extensibility. We made it clear that there may appear unforeseen requirements of support for new user or device types, and therefore a well-defined methodology should exist in order to minimize the effort and the time consumption wasted by the developer to extend the application according to the new requirement. Thus, we introduced the idea of defining the influence of the new characteristic on the models in the form of an adaptable task template. After that, we justified the need for extending dialog graphs with the possibility of having implicit transitions between the different views, as well as the need to extend the CTT notation with some new types which are in fact subcategorization of the already existing types. We made it clear that those new types will enable automating the adaptation of the application according to newly introduced characteristics and corresponding templates provided by the designer. The new dialog graph's implicit transition types as well as the new CTT task types have been presented. As we strive to automate the whole task model to user interface transformation process, we presented heuristics that have to be followed in order to derive dialog graphs out of the fine-grained task models. In order to highlight the contribution of the suggested extensions, we provided a brief example where we took benefit of the usage of the new CTT task types and implicit transitions. Our ultimate goal is to achieve individualized tailored user interfaces which serve every individual in the environment depending on his/her own profile. An evaluation of the resulting user interfaces is also planned.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

REFERENCES

Alexander, C. (1977). *A Pattern Language: Towns, Buildings, Construction*. Center for Environmental Structure Series. Oxford University Press, USA.

Annett, J. and Duncan, K. (1967). Task analysis and training design, occupational psychology. volume 411, pages 211–221.

Bergh, J. V. d. and Coninx, K. (2004). Contextual concurtasktrees: Integrating dynamic contexts in task based design. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, PERCOMW '04, pages 13–, Washington, DC, USA. IEEE Computer Society.

Bobick, A. F., Intille, S. S., Davis, J. W., Baird, F., Pinhanez, C. S., Campbell, L. W., Ivanov, Y. A., Schtte, A., and Wilson, A. (1999). The kidsroom: Perceptually-based interactive and immersive story environment. In *PRESENCE*, pages 367–391.

Borchers, J. (2001). *A Pattern Approach to Interaction Design*. John Wiley & Sons, Inc., New York, NY, USA.

Breedvelt-Schouten, I.M., P. F. and Severijns, C. (1997). Reusable structures in task models. Proceedings of DSV-IS.

Caffiau, S., Girard, P., Scapin, D., and Guittet, L. (2007). Generating interactive applications from task models: a hard challenge. TAMODIA'07, pages 267–272, Berlin, Heidelberg. Springer-Verlag.

Card, S. K., Newell, A., and Moran, T. P. (2000). *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.

Cook, D. and Das, S. (2004). *Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience.

Das, S. K., Cook, D. J., Bhattacharya, A., Heierman, E. O., and yun Lin, T. (2002). The role of prediction algorithms in the mavhome smart home architecture. *IEEE Wireless Communications*, 9:77–84.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Hartson, H. R. and Gray, P. D. (1992). Temporal aspects of tasks in the user action notation. *Hum.-Comput. Interact.*, 7(1):1–45.

Ju, W. and Leifer, L. (2008). The design of implicit interactions: Making interactive systems less obnoxious.

Kato, K. and Ogawa, K. (1993). Task analysis method using the goms model with grouping. In Salvendy, G. and Smith, M. J., editors, *HCI (2)*, pages 891–896. Elsevier.

Kirste, T., Aarts, E., and Encarnacao, J. L. (2001). Smart kindergarten: sensor-based wireless networks for smart developmental problem-solving environments. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, MobiCom '01, pages 132–138. Springer.

Klug, T. and Kangasharju, J. (2005). Executable task models. In *Proceedings of the 4th international workshop on Task models and diagrams*, TAMODIA '05, pages 119–122, New York, NY, USA. ACM.

Limbourg, Q. and Vanderdonckt, J. (2003). Comparing task models for user interface design. In *The Handbook of Task Analysis for Human-Computer Interaction*. Lawrence Erlbaum Associates.

Luyten, K., Clerckx, T., Coninx, K., and Vanderdonckt, J. (2003). Derivation of a dialog model from a task model by activity chain extraction. volume 2844, pages 203–217. DSV-IS 2003 : design, specification and verification - interactive systems.

Nijholt, A., Akker, R. O. D., and Heylen, D. (2004). Meetings and meeting modeling in smart surroundings. In *Social Intelligence Design. Proceedings Third International Workshop*, pages 145–158. ISBN.

Paterno, F. (2000). Model-based design of interactive applications. *Intelligence*, 11(4):26–38.

Paterno, F. (2001). Task models in interactive software systems. In *In Handbook Of Software Engineering And Knowledge*. World Scientific Publishing Co.

Paternò, F., Mancini, C., and Meniconi, S. (1997). Concurtasktrees: A diagrammatic notation for specifying task models. In *Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction*, INTERACT '97, pages 362–369, London, UK, UK. Chapman & Hall, Ltd.

Paterno, F. and Santoro, C. (2002). One model, many interfaces. In Kolski, C. and Vanderdonckt, J., editors, *CADUI*, pages 143–154. Kluwer.

Radeke, F. (2007). Pattern-driven model-based user-interface development. Master Thesis, University of Rostock.

Sara, B. (2009). *Smart Sensors For Interoperable Smart Environment*. PhD thesis, University of Bologna.

Schlungbaum, E. and Elwert, T. (1996). Dialogue graphs- a formal and visual specification technique for dialogue modelling.

Schmidt, A. (2000). Implicit human computer interaction through context. volume 4, pages 191–199. Springer-Verlag.

Sousa, K., Mendonça, H., and Vanderdonckt, J. (2010). A rule-based approach for model management in a user interface – business alignment framework. In *Proceedings of the 8th international conference on Task Models and Diagrams for User Interface Design*, TAMODIA'09, pages 1–14, Berlin, Heidelberg. Springer-Verlag.

Srivastava, M., Muntz, R., and Potkonjak, M. (2001). Smart kindergarten: sensor-based wireless networks for smart developmental problem-solving environments. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, MobiCom '01, pages 132–138, New York, NY, USA. ACM.

Waibel, A., Schultz, T., Bett, M., Denecke, M., Malkin, R., Rogina, I., Stiefelhagen, R., and Yang, J. (2003). Smart: The smart meeting room task at isl. In *in Acoustics, Speech, and Signal Processing (ICASSP '03). 2003: IEEE*, pages 752–755.

Weiser, M. (1999). The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11.

Wolff, A. and Forbrig, P. (2009). Deriving user interfaces from task models. In *Proceedings of the Model Driven Development of Advanced User Interfaces, MDDAUI'09*.

Wolff, A., Forbrig, P., and Reichart, D. (2005). Tool support for model-based generation of advanced user interfaces. In *The Unified Modeling Language*.

Wurdel, M., Forbrig, P., Radhakrishnan, T., and Sinnig, D. (2007). Patterns for task- and dialog-modeling.

Wurdel, M., Sinnig, D., and Forbrig, P. (2008a). Ctml: Domain and task modeling for collaborative environments.

Wurdel, M., Sinnig, D., and Forbrig, P. (2008b). Interactive systems. design, specification, and verification. chapter Task Model Refinement with Meta Operators, pages 300–305. Springer-Verlag.

Zaki, M., Bruening, J., and Forbrig, P. (2011). Towards contextual task patterns for smart meeting rooms. In *Pervasive and Embedded Computing and Communication Systems*.

Zaki, M. and Forbrig, P. (2011). User-oriented accessibility patterns for smart environments. In *Proceedings of the 14th international conference on Human-computer interaction: design and development approaches - Volume Part I*, HCII'11, pages 319–327. Springer-Verlag.