Prototyping of Multimodal Interactions for Smart Environments based on Task Models

Sebastian Feuerstack, Marco Blumendorf, Sahin Albayrak

DAI-Labor
Technische Universität Berlin
Secretary TEL 14, Ernst-Reuter Platz 7
D-10587 Berlin, Germany
{Sebastian.Feuerstack, Marco.Blumendorf,Sahin.Albayrak}@dai-labor.de

Abstract. Smart environments offer interconnected sensors, devices, and appliances that can be considered for interaction to substantially extend the potentially available modality mix. This promises a more natural and situation aware human computer interaction. Technical challenges and differences in interaction principles for distinct modalities restrict multimodal systems to specialized systems that support specific situations only. To overcome these limitations enabling an easier integration of new modalities to enhance interaction in smart environments, we propose a task-based notation that can be interpreted at runtime. The notation supports evolutionary prototyping of new interaction styles for already existing interactive systems. We eliminate the gap between design- and runtime, since support for additional modalities can be prototyped at runtime to an already existing interactive system.

1 Introduction

The change from single computing systems towards smart environments connecting complex networks of devices leads to new requirements for applications and human-computer interaction. A smart environment extends the amount of potential interaction devices from a small amount of isolated utilized devices like for instance a PDA or a smart phone to combinations of devices that enable addressing a mixture of modalities. Together with various interconnected sensors that make the environment aware of the user, interactions can be embedded in the environment: Not a specific device, but the whole environment can be used as the interface.

Whereas model-based approaches introduce various abstractions to design interactive systems, each enabling a comprehensive view of the whole application, they are currently focused on analysis- and design support. Thus, these approaches require the developer to consider all combinations of devices, which the interactive system should be able to adapt to during the development process. Each new mix of devices and modalities requires going through most of these design models again to compile a new user interface.

This paper presents our approach to prototype multimodal interactions in smart environments. It extends an approach that we have presented earlier [3] by supporting prototyping of additional modalities that have not been considered during

development of the application. Realizing such an approach requires two problems to be solved:

- A *notation* that is comprehensive enough to be interpreted at runtime while also providing the application designer with an overview of the whole application, because adding new multimodal interactions often affects the complete interaction flow of the application. The notation should support prototyping of new interactions without changing the original application.
- *Runtime interpretation of the user interface model with support for model synchronization*: The runtime environment must reflect commonly used design time models describing several abstract views of an interactive application and has to support mappings to the runtime environment to eliminate the gap between design and runtime. Every modification to the models that is done during prototyping has to be instantly propagated to be synchronized with all other models.

This paper focuses on the notation that we can interpret in our runtime environment that we have successfully implemented. Instead of presenting the runtime architecture in greater detail we concentrate on examples to discuss the most important aspect, the synchronization requirements for such a runtime environment. The paper is structured as follows: The next section starts with discussing the actual requirements and issues for prototyping multimodal applications in smart environments. Based on this discussion we propose a task model notation that enables a model-based evolutionary prototyping at runtime and specially addresses synchronization problems that occur when prototyping new forms of interactions to a running interactive application. To evaluate our approach we describe the Smart Home Energy Assistant (SHEA) that we have implemented with our model-based runtime environment (MASP[1]) [3] and enhance the SHEA with a new interaction device, a gesture and orientation aware remote control to illustrate our approach. Section 3 discusses the related work that has been considered. Finally, section 4 concludes the paper with a short evaluation of our approach and outlines future work.

2 A task modeling notation supporting smart environments

The runtime interpretation of task models requires a notation that specifies the interactive system detailed enough to be executed without further information. Thus it requires integrating static concepts with a temporal description. We found the ConcurTaskTree (CTT) notation [10] a good starting point as it allows a user-centric design enabling a step by step abstract to concrete modeling using task trees, which eases prototyping. The notation also allows focusing on the user's goals by defining sequences of tasks using a LOTUS-based temporal specification.

[1] MASP - Multi-Access Service Platform

However, to interpret CTT-based task trees generating user interfaces for smart environments at runtime, some changes to CTT are required:

- CTT abstracts from the functional core of the system, as for analyzing interactive systems it is focused to the interaction with the user. In smart environments continuously acquired sensor data or the physical control of appliances can influence the interaction with the user by triggering or disabling user interaction tasks.
- CTT allows specifying objects that are required for task performance but does not provide the possibility to explicitly model the object usage by including it into its graphical notation. From a designer's perspective an explicitly modeled object usage enables consistency and feasibility checks of the concept distribution. In smart environments tasks can be extensively distributed and the object usage enables to identify those tasks that are relevant for continuous updates of sensor data or synchronizes tasks that are interconnected by referring to the same domain objects.
- The notation is mainly targeted to model interactions that are independent from any modality. Modeling modality specific interactions is limited to hide tasks for devices with limited capabilities. Whereas CTT abstracts from how a tasks is performed by a specific device, decision nodes [2] can be used to design device specific task realizations. To our knowledge modeling interactions involving simultaneous usage of more than one device using the task model is not possible by using decision nodes, since decision nodes describe a choice and are not addressing simultaneous usage of more than one modality. Further on interdependencies between the used devices to implement complementary or redundant relations [8] between the modalities cannot be addressed.

In the following sections we describe how we are addressing these issues by introducing changes to the CTT notation to support modeling multimodal interaction in smart environments. First we describe the way to refer to the functional core of an interactive system by changing the task types; secondly we enhance the notation to explicitly annotate the domain concepts as well as the object flow through the task tree. Finally we introduce our approach to prototype the simultaneous usage of modalities using interconnected task trees.

2.1 Referencing the functional core from a task tree

Different to CTT we distinguish two types of interaction tasks: output interaction tasks (OIT) and input interaction tasks (IIT). While OITs require no human intervention but present information to the user until they become disabled by another task, IITs require human intervention like data input. Additionally to this understanding of interaction tasks we use a different interpretation of "application tasks" since modeling the information exchange with the backend system is crucial to be able to use the task tree definition for the user interface creation at runtime. To enable a loose coupling of the smart environments infrastructure to the user interface and an easier inclusion of backend services, we define application tasks as solely

executed by the application without user's intervention, not offering a presentation to the user.

2.2 Annotating domain concepts and modeling the object flow

Utilizing task trees as the workflow definition of a user interface at runtime also requires a stronger and more detailed connection of task and domain concepts. Such a connection on the one hand allows the identification of inaccurate domain abstractions not conforming to the task abstraction as well as the review of the distribution of concepts over the whole task tree and on the other hand provides the basis to derive meaningful user interfaces based on the task tree definition. While CTT already allows the annotation of objects and the marking of an LOTUS operator as "information passing" [9], this approach does not completely reflect the requirements to model interactive systems embedded into smart environments. Such systems require modeling of the object flow throughout the complete task tree, which considers the object synchronization not only between two tasks but between all tasks that are enabled and are using the same domain concepts. Following the approach of Klug et al. [5] we annotate the required objects as well as the access type directly to each task of the task tree allowing a fine grained modeling of the object flow.
An abridged version of the task model of the SHEA application we have implemented is depicted in figure 1. The application discovers all appliances, which we have installed in our ambient test-bed and presents an overview of all appliances. By selecting an appliance the user can control and inspect past energy consumptions as well as extrapolations of future consumptions by issuing voice commands or using a wall-mounted touch display. Like depicted in figure 1, the access type is limited to read (R), modify (M) or create (C) operations. Introducing the declaration (D) of each object before the first usage also allows the definition of a scope for each object. Similar to a programming language each object has to be declared before it can be created and only can be modified or read after it has been successfully created. Under each task's name, we annotate the domain objects type in square brackets (by referencing a class name) together with the domain object name, whose attributes can be accessed by using a dot followed by the attribute name.

During interpretation of the task tree at runtime the domain object annotation is used to synchronize the enabled tasks objects, which are actually performed. Thus, if one task modifies a domain object and another task that is enabled concurrently has read access to the same domain object, the task receives the object's modifications instantly. Further on, the declaration of objects enables the runtime environment to keep track of objects and to remove objects that are no longer required as the user continues the interaction.

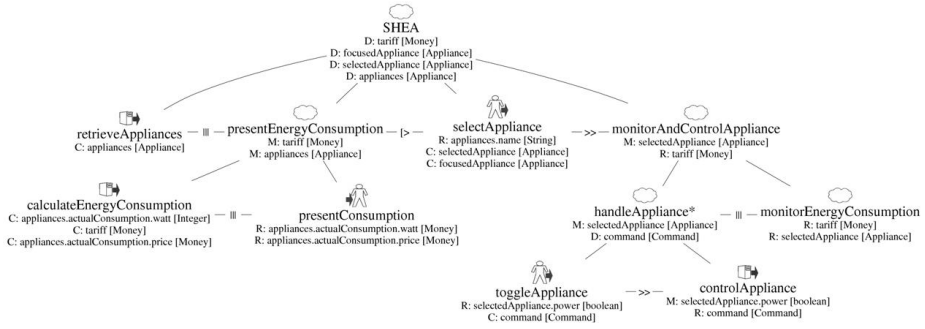

Fig. 1. Abridged task tree of the SHEA application

Inaccurate domain abstractions that do not conform to the task abstraction can be identified easily by checking the cluttering and granularity of the object annotations. Furthermore basic automatic consistency checks (i.e. create before modify and read) across multiple subtasks are possible, because declarations as well as all defined operations are also summarized in the next higher level of abstraction. The designer can further partition and detail domain concepts in sub-trees together with the tasks performing on the concepts. This ensures that the resulting interactive application is not switching between several concepts too often, which make the application complicated to use, since it raises the cognitive workload of the user. If the same concepts are spread over the whole tree remodeling the task tree should be considered, regarding two aspects. On the one hand the conceptual abstractions should be reviewed and it should be checked, whether these concepts can be detailed conforming to the associated task abstractions. On the other hand the domain object distribution should be solved by moving tasks that are performing on the same domain objects together into the same sub-tree.

2.3 Interconnecting tasks and objects to prototype the simultaneous usage of modalities

Prototyping of additional interaction capabilities for an already existing interactive application is done by designing a new modality specific task tree. This tree addresses the specific tasks that are describing a new way of interaction that the additional interaction device should support. Modality specific trees are interpreted in parallel to the application task tree and are related to the application task tree by referring to some of the application task tree's objects and tasks.

Figure 2 depicts a modality specific task tree we have prototyped for the SHEA application that describes a remote control that is aware of the direction it points to and is able to detect gestures that are interpreted as appliance control commands. The remote requires the user to point to an appliance first, and then waits that the user presses the remote's button, which starts the gesture detection and initiates a command for the selected appliance that is issued after the button has been released by the user.

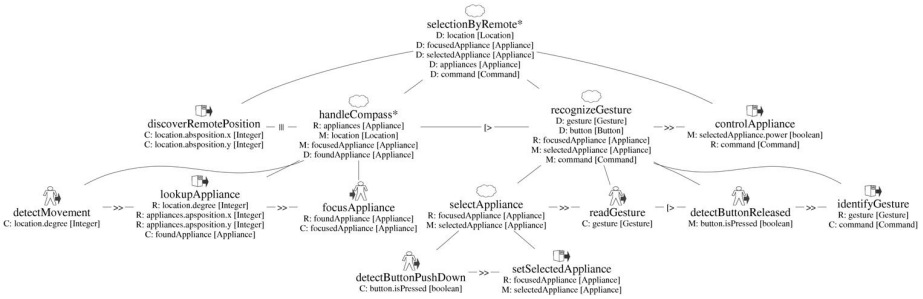

Fig. 2. A modality specific task tree that describes a remote with support for gesture and orientation detection to control interactive applications.

The modality specific tree can be interconnected by the designer with the application tree using two synchronization methods: On the one hand domain objects of the application task tree can be synchronized with the modality specific task tree. This is specified by annotating the same domain objects names to the modality specific task tree. For instance, to interconnect the object that the remote is pointing to (see figure 2: *focusedAppliance*) with a presentation of the focus in the SHEA task tree (in the *selectAppliance* task of figure 1). Further on, all discovered appliances (stored by the appliances domain object) as well as the actual selected appliance (*selectedAppliance* object) get synchronized between the remote and the SHEA application.

On the other hand tasks of the modality specific tree can be connected to tasks of the application task tree. In our example we use task synchronization when the user has to choose the application he wants to control (in the *selectAppliance* task). Since both trees are started simultaneously, the first task that the user is required to perform in the SHEA application is to select an appliance to monitor its energy consumption history. Following the design of the application task tree this has to be done by choosing one appliance of a list of appliances. Using the remote the user just needs to point to the appliance (*handleCompass*) and press the remote's button to select it to perform the *selectAppliance* task. We interconnect both *selectAppliance* tasks using a directional connection, which is only enabled if the connected application task's trees task is enabled and does not require changes to the application task tree. The remote can be used independently but if both trees are in the same state, requiring the user to select an appliances both *selectAppliance* tasks can be either performed by using the remote or the touch display originally supported by the application task tree.

In our current implementation we have implemented support for interconnecting IIT and support every temporal operator of the CTT notation for interconnected tasks. Interconnecting application tasks is supported as well (see *controlAppliance* task in both figures) but only describes that a call to the functional core has to be done once only. The modality specific tree should follow the basic interaction flow of the application task tree since this eases the complete integration of the prototyped interaction with the new modality into the application later on. Following a prototyping approach, not the whole application is required to be supported by the modality specific task tree. Thus, in the beginning only parts can be covered by the new modality and can be continuously evolved during the prototyping process.

The order of all interconnected tasks must remain the same for both, the application task tree and the modality specific task tree. In our actual implementation the remote of figure 2 cannot be designed to support initiating the control command before selecting an appliance, since this will break the design of the SHEA application. But requiring the same fixed sequence of interconnected tasks for all modality specific trees has the benefit, that the modality specific task tree can be merged with the application task tree after the prototyping process has been finished.

3 Related Work

Research in the area of model-based user interface development mainly concentrates on the creation of models and tools suitable to design models and then derive final user interfaces from these models following a transformational approach following the ideas of the CAMELEON reference framework [1]. Current prototyping approaches following a model-based development process support high- as well as low fidelity prototyping [12] but are limited to design time or are focusing on simulation of the targeted environment using virtual reality technologies [4].

Pribeanu et al. [11] have identified three ways to relate context-sensitive parts to context-insensitive parts of task trees: Firstly, by following a monolithic approach by specifying both parts in one tree. Secondly, by following a separation approach both trees can be connected using decision nodes. Finally, by following a graph-oriented approach that uses general arcs to connect both trees. Clerckx et al. [2] follow the separation approach introducing decision tasks as a modified CTT notation to support context-sensitive user interfaces. In the separation approach continuous context switching at any time on behalf of the user is limited to happen only before a separation task. After a decision has been taken the user or the system, context switching without adding special mappings is not possible and ends up in context switching problems [7]. This is why we rely on separate task trees that are linked to the main applications task tree to prototype various contexts of thus follow a graph-oriented approach. Lewandowski et al. [6] propose an approach for task model assembling based on tree algebra operators to automatically merge task trees. Their composition mechanism can be used to merge the modality-independent application tree with the prototyped modality specific task trees to derive a monolithic task tree.

4 Conclusion and Future Work

Supporting multimodal interaction in smart environments requires extended methods to model applications and their user interfaces. In this paper we present an approach to support the prototyping of additional modalities to already running interactive systems based on a modified Concurrent Task Tree notation. The notation explicitly includes the annotation of domain concepts to make the task designer aware of concept distribution and supports modeling actions triggered by context changes through the use of application tasks. We are currently working on formalizing and improving our notation by supporting the tree concatenation approach [6] to merge a

successfully prototyped modality back to the main application task tree. In addition we are currently experimenting with prototyping a combined gaze direction and voice control for the SHEA application to further improve our user interfaces runtime environment for smart environments (MASP) in our Ambient Living Test-bed, set up as part of the Service Centric Home[2] project at the DAI-Labor of the Technische Universität Berlin.

References

1. Calvary, G. et all: A unifying reference framework for multi-target user interfaces. In: Interacting with Computers, Vol. 15, No. 3. pp. 289-308, (June 2003).
2. Clerckx, T.; den Bergh, J. V. & Coninx, K.: Modeling Multi-Level Context Influence on the User Interface. pp. 57-61 (2006).
3. Feuerstack, S.; Blumendorf, M. & Albayrak, S. (2006), Bridging the Gap between Model and Design of User Interfaces, in 'Proccedings of Informatik 2006', Dresden, 2006, pp. 131-137
4. Gonzalez, J.M., Vanderdonckt, J., Arteaga, J.M.: A Method for Developing 3D User Interfaces of Information Systems. In: Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006 (Bucharest, 6-8 June 2006), Chapter 7, Springer-Verlag, Berlin, 2006, pp. 85-100
5. T. Klug, J. Kangasharju: Executable task models. In: Proceedings of the 4th international Workshop on Task Models and Diagrams, pp. 119-122. TAMODIA '05, ACM Press.
6. Lewandowski, A., Lepreux, S., Bourguin, G.: Tasks models merging for high-level component composition.In: Proc. of HCI International 2007, 2007
7. Luyten, K.; Vandervelpen, C.; den Bergh, J. V. & Coninx: Context-sensitive User Interfaces for Ambient Environments: Design, Development and Deployment. In_ Mobile Computing and Ambient Intelligence: The Challenge of Multimedia, 2005
8. Nigay, L. & Coutaz, J. Multifeature Systems: The CARE Properties and Their Impact on Software Design Intelligence and Multimodality in Multimedia Interfaces, 1997
9. Mori, G., Paternò F., Santoro, C.: Design and Development of Multi-Device User Interfaces through Multiple Logical Descriptions. In: IEEE Transactions on Software Engineering, 2004.
10. Paterno, F.: Model-based Design and Evaluation of Interactive Applications. Springer Verlag. Berlin 1999.
11. Pribeanu, C.; Limbourg, Q. & Vanderdonckt, J.: Task Modelling for Context-Sensitive User Interfaces. In: Lecture Notes in Computer Science, 2001, 2220
12. Sottet, J., Calvary, G., Favre, J., Coutaz, J. & Demeure, A.: Towards Mappings and Models Transformations for Consistency of Plastic User Interfaces. In: Proceedings of CHI 2006 Workshop "The Many Faces of Consistency in Cross-Platform Design", 2006

[2] http://www.sercho.de