Models and Patterns for Smart Environments

Peter Forbrig
University of Rostock, Department of Computer Science
Albert-Einstein-Str. 22
D-18051 Rostock, Germany
+49 381 498 7620
peter.forbrig@uni-rostock.de

Maik Wurdel, Michael Zaki
University of Rostock, Department of Computer Science
Albert-Einstein-Str. 22
D-18051 Rostock, Germany
+49 381 498 7431
michael.zaki@uni-rostock.de

ABSTRACT
In a given smart meeting room, several users are supposed to cooperate together while employing static and dynamic heterogeneous devices. The goal of such environments is to deliver proper assistance to the users while performing their tasks. Thus, task models are an appropriate starting point for those environments. Those models give the developers the opportunity to focus on the users and their tasks. Tasks are not independent from available tool, locations and acting persons. Therefore, other models have to be developed and linked to the task model in order to truly illustrate how the tasks are executed in those environments. The paper discusses the application of the language CTML that was designed for this purpose. Furthermore, the usage of patterns for supporting the development of models for smart environments will be discussed.

Keywords
Smart environment, HCI, task model, context, goal pattern, task pattern

INTRODUCTION
The idea of ubiquitous computing goes back to Marc Weiser. According to Weiser's vision [11], devices are weaving themselves into everyday life, allowing people to fully concentrate on performing their tasks, while hiding their existence and complexity. A smart environment (SE) tries to analyze user behavior and tries to provide appropriate assistance. Within the context of a meeting scenario the presenter should concentrate on the talk, while the SE is responsible for offering convenient assistance by adjusting the projector, loading the necessary files and capturing audiovisual data for meeting documentation if needed. In the best case no direct interaction is necessary. Implicit interaction like going to the presentation area is enough to present the slides of the speaker. Experiences show that [5] that the quality of support can be increased if some information is given to the system. Most important are the tasks the users want to perform within the environment.

Task models are an appropriate starting point for interactive processes development. The application of task models for smart environments is discussed in [4] We will shortly introduce the concept of task models and an own collaborative task-modeling language will be presented. Afterwards we introduce our ideas of using patterns.

MODELLING COOPERATION WITH TASK TREES
Currently CTT [7] is one of the most referred notations for task models. Tasks are arranged hierarchically, where more complex tasks are decomposed into simpler sub-tasks. CTT distinguishes between several task types, which are represented by the icon representing the task node. There are abstract tasks, which are further decomposable into combinations of the other task types including interaction, application and user tasks (see Fig. 1 for an overview of the available task types). The task type denotes the responsibility of execution (human, machine, interaction, cooperation with human).

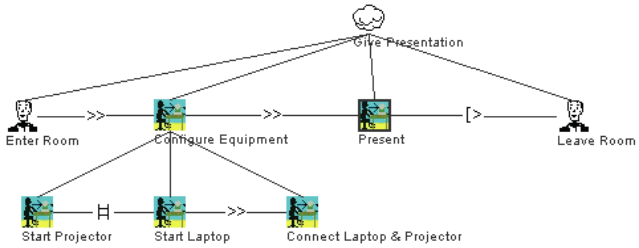

Figure 1 Task model for giving a presentation

Sibling tasks are connected by binary temporal operators. However, unary operators exist that are related to one task only. A complete listing and explanation of the CTT operators can be found in [7]. Operators have precedence orders. These orders are important for interpreting different operators in the same level. The priority of the interleaving (|=|) operator is higher than the enabling operator (>>). The iteration is an example of a unary operator. An example of a CTT model within the context of a smart environment for giving a presentation is shown in Figure 1. It provides the task of giving a presentation. The abstract root task "Give Presentation" is decomposed into four child tasks. The left tasks on the second level of abstraction are connected with the enabling operator (>>) in order to specify that one task has to be performed before the other one can start (e.g., the interactive task "Configure Equipment" can only be performed after having executed the human task "Enter Room"). "Leave Room" can be performed at any time due to the deactivation operator ([>) resulting in a prematurely

abortion of the currently running task. Furthermore, the task "Configure Equipment" is decomposed into the subtasks "Start Projector", "Start Laptop" and "Connect Laptop & Projector". The third task can only be executed after the first two tasks were performed.

Because of lack of space the model does not specify the details of the presentation. Models can be of course as detailed as necessary.

COLLABORATVE TASK MODELING LANGUAGE

In conjunction with modeling efforts in a smart environment the collaborative task modeling language (CTML) was developed. Despite that the idea was originated in the context of smart environments, it seems to be applicable in a broader range (e.g. Stakeholder-driven process management can be supported in this way). We will shortly discuss the fundamental assumptions and the most significant features of the language.

The design of CTML is based on four fundamental assumptions:

I. **Role-based Modeling**. In limited and well-defined domains the behavior of an actor can be approximated through her role.

II. **Hierarchical Decomposition and Temporal Ordering**. The behavior of each role can be adequately expressed by an associated collaborative task expression.

III. **Causal Modeling**. The execution of tasks may depend on the current state of the environment (defined as the accumulation of the states of all available objects) and in turn may lead to a state modification.

IV. **Individual and Team Modeling**. The execution of individual user tasks may contribute to a higher level team task

Based on these assumptions a collaborative task model is specified in a two-folded manner:

1. Cooperation Model.

   Specifies the structural and behavior properties of the model.

2. Configuration(s).

   Holds runtime information (like initial state, assignment) and simulation / animation configurations.

A cooperation model is presented in Figure 2. Model entities are represented by elements in the inner circle (post fixed with "-1"). Diagrams outside of the inner circle provide more detailed specifications of the corresponding entities (post fixed with "-2"). A specification of a model consists of specifications of roles (e.g., A-1), devices (e.g., B-1), a location (C-1), a domain (D-1) and a team (E-1).

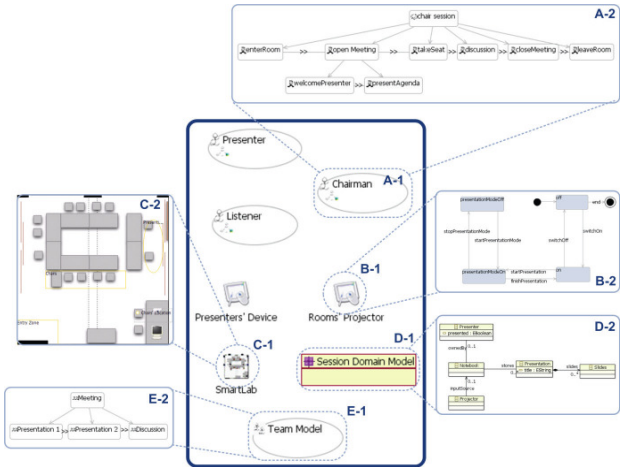

Figure 2 Schematic Cooperation Model for a Meeting

In our following discussion we will focus on roles and their models only. Roles categorize users that have the same capability, responsibility, experience and limitations according to the domain. Thus roles can be considered as abstractions of users sharing the same characteristics. In software engineering roles are often called actors. The potential actions a user is able to perform are determined by his role(s). In CTML a role is associated with a task model (A-2) that is visually represented by a task tree in a CTT-like notation.

CTML allows the dynamic change of roles during runtime, which is not very common in other modeling approaches.

Let us assume that Sheldon acts as Chairman and Leonard acts as Presenter in our smart meeting room. Additionally, there is Penny. She fulfills the roles Presenter and Listener as depicted in Fig. 3.

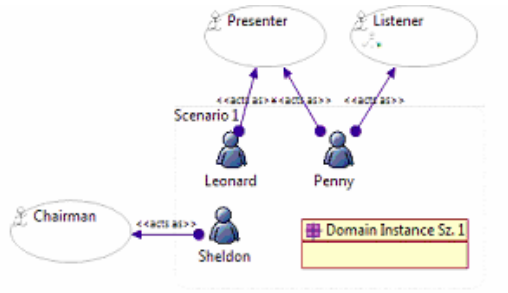

Figure 3 Specific Meeting Configuration

The configuration assigns persons to roles. Additionally, it can be specified that Penny first acts as a presenter and later fulfills the role of a listener. This can be expressed by "Presenter >> Listener" for Penny. A configuration is sometimes also considered as a scenario for which the cooperation model is used.

Sometimes temporal relations are not expressive enough to specify the real constrains between different tasks. This was the reason for introducing textual specifications into

CTML (like OCL [8] for UML). Such specifications allow constrains that involve devices, locations and all other model elements. They are used to specify preconditions and effects of tasks using an OCL-like syntax. For the role 'Chairman' and the role 'Presenter' the preconditions and effects shown in Table 2 make sense.

Table 1 Examples for preconditions and effects

Role	Task	Precondition
1 Presenter	Start presentation	Chairman.oneInstance.AnnounceTalk
2 Chairman	Announce discussion	Presenter.allInstances.EndPresentation
Role	**Task**	**Effect**
1 Presenter	End resentation	self.presented = true
2 Chairman	Announce discussion	Notebook.allInstances.switchOff

A presenter is only allowed to start his presentation after the chairman has announced it (precondition 1). Precondition 2 states that a chairman can only announce a discussion if all presenters have finished their presentations. It might be a little bit difficult to create such kind of specifications but they have the advantage of being readable to some extent. For the expressiveness of such specifications quantifiers are very important. They allow specifying the number of actors or devices (one or all).

After a Presenter ended his talk the corresponding attribute is set to true (effect 1). After the Chairman has opened the general discussion all notebooks in the room are switched off (effect 2).

It is possible to specify activities taking place in a smart environment in a precise way. However, it is sometimes a burden to develop such a specification. The modeling process is complex and time consuming. A promising idea would be to overcome this problem by using existing specifications to build new ones.

Patterns have proved [5] to be a good tool to represent knowledge in software design. They spread through computer science domain despite the fact that they were first discussed in architecture [1]. Additionally, many approaches take benefit of the usage of patterns in the HCI area [21]. Breedvelt-Schouten et al. [3] introduced task patterns that inspired our work. Sinnig [9] provided generic task patterns to be able to adapt a pattern to the context of use.

In a given smart environment numerous actors try to achieve a common goal that can be characterized as team goal. For the meeting room example, the ultimate goal is the efficient exchange of information among the actors in the room. Every task executed by an actor in its role is in a way a contribution to the team goal. It is a step towards this goal. Additionally, the task helps to reach the own individual goal (e.g. to make a good presentation).

A first step to develop patterns in the context of smart meeting rooms was to identify possible team goals (a certain state that the team wants to reach). First results were presented in [25] by providing six abstract team goals. These goals were (I) conference session performed, (II) lecture given, (III) work defended, (IV) topic discussed, (V) debate managed and (VI) video watched.

In the meantime some further patterns were identified. Figure 4 presents one of those patterns in a simplified way. It is a team pattern for discussing phenomena of the climate that was identified by observing meetings in a research institute.

Usually during meetings at this institute there is first a general presentation. Later on participants split into two subgroups and discuss some pictures and data. in two subgroups and at the end the combined results from both groups are presented to the whole plenum.

Unlike the former task patterns approaches it is our goal to integrate the so-called "forcing context" into the pattern specification. The forcing context describes the set of environmental preconditions that have to be fulfilled in order to execute the tasks within the pattern and the set of post-conditions expressing the influence of the execution of those tasks on the state of the environment.

smart environments. Moreover, the main trend nowadays is the design of universally accessible applications.

The pattern description consists of an ID, name, problem, situation, solution, diagram, adaptation variables and referenced patterns. In this paper we will concentrate on the illustrations (in the diagram section of the pattern) that are provided within the pattern example of Figure 5.

The diagram section of the pattern consists of three parts, the task hierarchy, the environmental dependencies and the visualization of the execution constrains.

Currently we are extending the CTML editor which provides an Eclipse-based IDE to build task models by our task pattern application tool. We truly believe that having this pattern library fostered by the tool, the developer will be offered a real assistance while modeling a given scenario in the context of smart environments.

CONCLUSIONS

In this paper we argued for a model-based approach for smart environments. We presented some details of our specification language CTML that allows specifying the tasks of different actors and the cooperation of a team. It is argued to split the specification into a cooperation model and a configuration model. The cooperation model specifies general knowledge of activities of a specific domain. This knowledge is long lasting. The configuration model has to be specified according to the current instance of a session. Who are the participants that take part and which roles do they play?

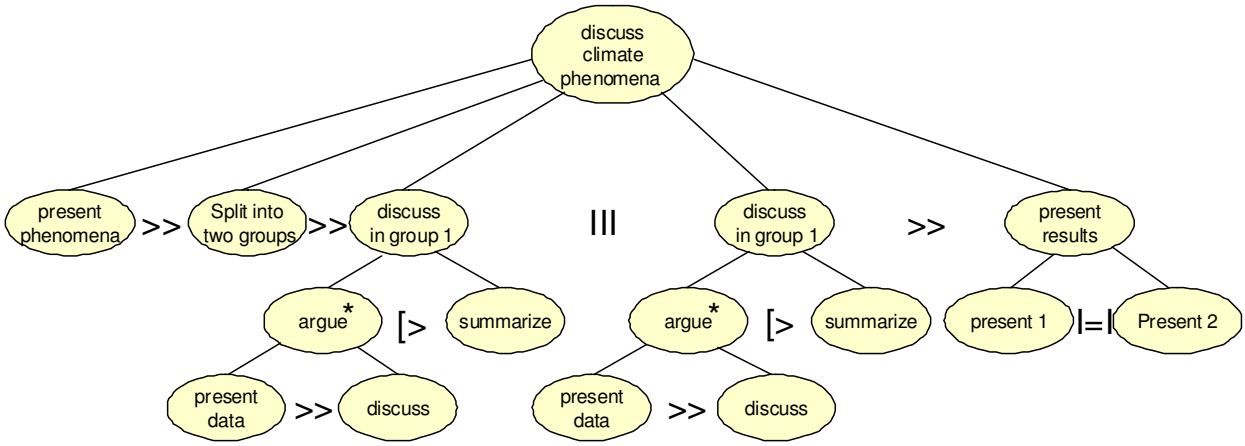

Figure 4 Fraction of a Team Pattern Example

Afterwards, we argued that the need to build the task models, the other environmental and domain models as well as the specification of the relations and constraints between all of the included models dramatically increase the complexity of the modeling process of those environment. Therefore, we suggested the usage of appropriate patterns which aim to provide convenient support to the developer while modeling her scenario.

The animation of our models can support the Bayesian algorithms of a smart environment that try to infer next possible actions of the users based on the sensor data. On the other hand these algorithms can inform the animation of the models that certain preconditions of task are fulfilled.

REFERENCES

1. Alexander, C., Silverstien, M.: A Pattern Language. In: Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid F. King und Shlomo Angel (Eds.), Towns, Buildings, Construction, Oxford University Press, New York 1977, ISBN 0195019199
2. Blumendorf, M.: Multimodal Interaction in Smart Environments A Model-based Runtime System for Ubiquitous User Interfaces. Dissertation, Technische Universität Berlin, 2009.
3. Breedvelt-Schouten, I.M., Paterno, F., Severijns, C.: “Reusable Structures in Task Models”, 1997, In Proceedings of DSV-IS: 225-239.
4. Blumendorf, M., Lehmann,G.,Albayrak, S.: Bridging Models and Systems at Runtime To Build Adaptive User Interfaces. Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems. ISBN: 978-1-4503-0083-4 : 9-18.
5. Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley, 1995
6. Giersich, M., Forbrig, P., Fuchs, G., Kirste, T., Reichart, D., and Schumann, H.: Towards an Integrated Approach for Task Modeling and Human Behavior Recognition, Proc. HCII 2007, p. 1109-1118, ISBN: 978-3-540-73105-4.
7. Mori, G., F. Paternò and C. Santoro: CTTE: Support for Developing and Analyzing Task Models for Interactive System Design, IEEE Trans. Software Eng. 28(8), 2002, p. 797-813.
8. OCL: http://www.omg.org/technology/ documents/modeling_spec_catalog.htm#OCL.
9. Sinnig, D.: The Complexity of Patterns and Model-based Development, Computer Science. Montreal, Concordia University. (Thesis 2004).
10. Tidewell, J.: Interaction Design Patterns: Twelve Theses, PLoP’98, Monticello, Illinois, In. Proc. Conference on Pattern Languages of Programming, 1998.
11. Weiser, M.: The Computer for the 21st Century. Scientific American, 265, pp.94-104, 1991.
12. Wurdel, M., Sinnig, D., Forbrig, P.: CTML: Domain and Task Modeling for Collaborative Environments. Journal of Universal Computer Science 14, 2008, p. 3188-3201.
13. Zaki, M., Forbrig, P.: User-oriented Accessibility Patterns for Smart Environments. Springer Volume 6761/2011, 319-327, DOI: 10.1007/978-3-642-21602-2_35.
14. Zaki, M., Forbrig, P.: Towards a Pattern Language for Modeling Interactive Applications in Smart Meeting Rooms.