HCI-Task Models and Smart Environments

Maik Wurdel*, Stefan Propp*, and Peter Forbrig

University of Rostock, Department of Computer Science
Albert-Einstein-Str. 21, 18059 Rostock, Germany
{maik.wurdel, stefan.propp, peter.forbrig}@uni-rostock.de

Abstract The paper discusses the idea of using HCI-task models to support smart environments. It introduces a collaborative task modeling language CTML that allows the specification of collaboration and comprehensive dependencies in an OCL-like style. Additionally some ideas are presented that allow informing users and usability experts about the state of actors within smart environments. The paper provides the first results of a prototypical implementation.

Keywords. HCI, Task Models, Model-based Usability Evaluation

1. Introduction

In the domain of HCI task analysis and modeling is a mature research area. Task models are used to elicit requirements in early stages of development by describing how people achieve goals by performing a set of tasks. However, in recent years, task models have also been employed for system design. Exemplary in the research field of model-based user interface (UI) development task models serve as initial model for model-based processes. In contrast in the research field of smart environments HCI task models have only been used barely. From our point of view this fact is quite surprising because smart environments comprise a vast complexity in terms of task performance of users. A thorough understanding of the tasks users are executing within such environments is a precondition to deliver an appropriate assistance.

In this paper we focus on using task models in smart environments to, first, understand the envisioned assistance and, second, to track the task performance during runtime. This approach consists of two major components: (1) the collaborative task modeling language to model the behavior of actors within smart environments and (2) usability evaluation methods to provide usability experts with evaluation support and to inform actors about the current state of the system.

*Supported by a grant of the German National Research Foundation (DFG), Graduate School 1424, Multimodal Smart Appliance Ensembles for Mobile Applications (MuSAMA)

Please use the following format when citing this chapter:

Wurdel, M., Propp, S. and Forbrig, P., 2008, in IFIP International Federation for Information Processing, Volume 272; *Human-Computer Interaction Symposium*; Peter Forbrig, Fabio Paternò, Annelise Mark Pejtersen; (Boston: Springer), pp. 21–32.

Additionally we introduce our tool support which allows for modeling and simulation of collaborative tasks and their execution environment. Our simulation environment allows for interactively walk through the designed artifact, while conducting the usability evaluation.

The remainder of the paper is structured as follows: in Secion 2 we stress some background information to tasks and smart environments. Section 3 introduces our Collaborative Task Modelling Language (CTML) and the corresponding tool support which is followed by Section 4 where usability evaluation methods for smart environments are discussed. Finally we draw the conclusion and give an outlook for future research avenues.

2. Modeling Tasks in Smart Environments

Within smart environments tasks are barely carried out in isolation, but have to be synchronized with other users' tasks. Some tasks cannot be started while others are still in progress. To motivate our research we illustrate the challenges of smart environments concering task models by a scenario. Afterward we reiterate through existing approaches of task modeling and its employements.

The session chair Dr. Smith introduces herself and defines the topic of the session. Afterwards she gives the floor to the first speaker who sets up her equipment, the laptop switches to presentation mode and the speaker starts with the talk. During the presentation the audience accesses additional information related to the talk using their personal devices. While the meeting proceeds the personal devices provide guidance and offer related information according to the current talk and the meeting progress. The chairman interrupts the speaker since she overruns her time slot. The plenum is asked for some brief questions which are answered by the speaker. Eventually the chairman closes the talk and announces the next one. Subsequent talks are given in a simliar same manner.

We consider a smart environment as location where people are collaborating using a set of stationary and mobile devices. The devices are supposed to support the users' tasks which have to be performed to achieve a certain goal (like giving a talk). Addtionally interaction with the environment is performed in a much broader way (Shirehjini, 2007) then in desktop applications. The initiative can be expressed explicitly or implicitly. An implicit interaction is understood as an action not performed to interact with the environment but interpreted by the system. Ideally an implicit proactive meeting assistant for instance does not wait for an explicit user command, but senses movements and gestures of the user via sensors to derive the assumed user intention and automatically provides support for the expected next task.

Based on the introductionary scenario we can elicit the key characeteristic of smart environments from the view of task modeling: (1) A vast amount of potential tasks supported by a dynamic set of devices. (2) The temporal order of tasks

depends on the collaboration of actors within the environment. (3) The state of the smart environment (defined as composed state based on each device) can furthermore restrict or enable the execution of a certain set of tasks.

Diverse notations for task models have been introduced (GOMS, HTA, CTT, WTM (Bomsdorf, 2007; van Welie, 1998)). Even though they differ in terms of presentation, expressiveness, level of formality and granularity they all share the same following basic principle: tasks are arranged hierarchically representing the decomposition of tasks and tasks are performed to achieve a certain goal. The decomposition of tasks stops when an atomic level is reached: the action. It builds the fundamental execution unit.

The most common notation ConcurTaskTrees (CTT) supports, amongst others, the concept of temporal relations which restricts the valid sequences of tasks to achieve a certain goal. Another asset of this notation is its tool support: CTTE (Mori, 2002). Various extensions have been introduced: Examplarily in (Bomsdorf, 2007; Klug, 2005) an action is not seen as atomic anymore, but defined by a life cycle. This defines a task more precisely which is employed to trigger events. The first approach does not consider a temporal operator as state chart whereas the latter does not consider abortion or skipping of tasks.

Modeling cooperation of users in terms of task models has been addressed by CCTT (Collaborative ConcurTaskTrees) (Mori, 2002). Similar to the corporative task modeling language presented in this paper, CCTT uses a role-based approach. A CCTT specification consists of multiple task trees. One task tree for each involved user role and another as a "coordinator" that specifies the collaboration and global interaction between involved user roles.

Model-based usability evaluation approaches, like RemUSINE (Paterno, 2007), capture interaction events to derive the performed user interaction on an abstract task-based level. A trace of task events contains qualitative information about accomplished tasks, as well as quantitative measures about durations of fulfilled tasks. Analysis approaches comprise e.g. (Malý, 2007; Paterno, 2007). The suggested visualizations are based on a linear time-based scale. However, the visualization approach presented in this paper applies a semantic lens to focus on a certain period of time.

After reviewing existing approaches we introduce our specification language which comprises the characteristics of smart environments for task modeling.

3. CTML – the Collaborative Task Modeling Language

CTML is based on the idea that in limited and well-defined domains the behavior of an actor can be approximated through her role and, second, the behavior of each role can be adequately expressed by an associated collaborative task expression.

According to this statement we correspondingly define a collaborative task model as a tuple consisting of a *set of actors*, a *set of roles,* a *set of devices* and a

set of collaborative task expressions (one for each role) where each actor belongs to one or more role(s).

Definition 1: (Collaborative Task Model). A collaborative task model G is a tuple $G=\langle A,R,T,D,F,a,r,p\rangle$ where:

A,R,T,D are non empty sets of actors, roles and collaborative task expressions and devices. F is the set of features of the model consisting of elements of the following kind: $\langle key, value\rangle$

$a{:}A \rightarrow P(R)$ is a function that associates an actor with a set of roles.

$r{:}R \rightarrow T$ is a bijective function that associates a role with a task model.

$p{:}A \cup D \rightarrow F$ is a relation associating features to the actors and devices

Each collaborative task expression has the form of a task tree, where nodes are either tasks or temporal operators. Each task is attributed with a (unique) identifier, a precondition and an effect. Intuitively, the precondition defines a required state of the collaborative environment for executing the task, whereas an effect denotes the resulting state after having executed the task. Addtionally temporal operators restrict the potential sequences of task performance.

Definition 2: (Collaborative Task Expression). A collaborative task expression CTE is a tuple $CTE=\langle T,h\rangle$ where,

T is a non-empty set of tasks of the form $\langle id, precondition, effect\rangle$

$h{:}\ T \rightarrow List(T) \times Op$, with $Op=\{[],|=|, |||, |>, [>, >>, *, \#, \text{opt}\}$ is a function that maps a task t to an ordered list of tasks and a temporal operator.
The former represents the children of task t, whereas the latter denotes the execution order of the children according to the given definition in (Sinnig, 2007).

We say a collaborative task expression is *well formed* if the corresponding task tree is connected and free of cycles such that each task (except for the root task) has exactly one parent. Moreover we demand that if a task has more than two children it is associated with an n-ary operator. If a task has exactly two or one child(ren) it is associated with a binary or unary operator respectively. Leaf tasks are not related to a temporal operator by the function h. This definition results in a bi-parit graph whose vertexes are either of T or Op. The function h defines the edges of the graph.

In Fig. 1 a subset of the collaborative task model for the introductory example is given that was interactively created using the CTML Editor. For the sake of readability for each task only the hierarchical breakdown and temporal relations are shown in a CTT-like style. Our editor is able to present temporal relations as nodes or in the CTT-style. Preconditions and effects have been omitted for the example below. An overview of the entire specification is given in the lower left corner.

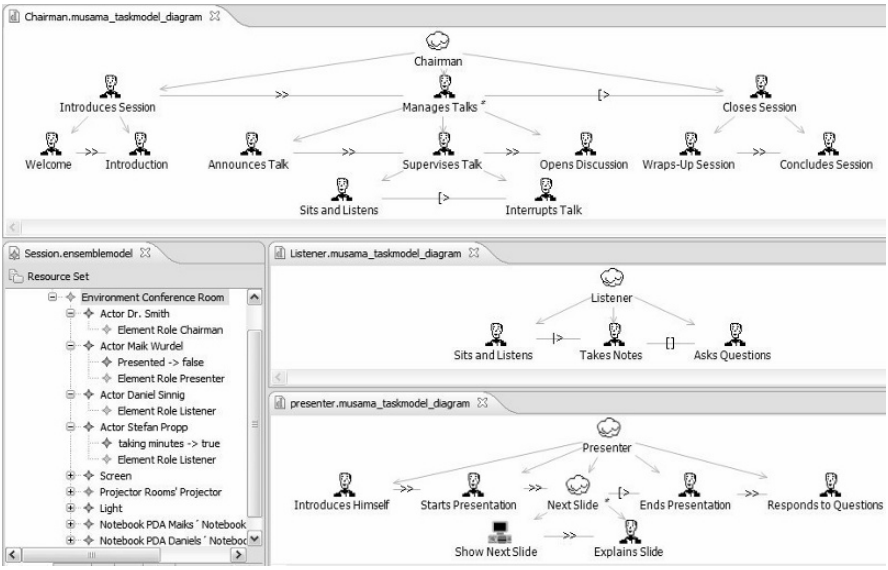

Fig. 1. Specification of the Example Using the CTML Editor

At runtime, for each active actor, an instance of the corresponding collaborative task expression (identified by the assigned role) is created. To implement the operational semantics of a CTML model all instance task expressions are translated to sets of communicating state charts. On the one hand task state charts are responsible for checking the precondition and manipulating the system to achieve its effect. On the other hand temporal operator state charts implement the semantics of its corresponding operator. In particular they mediate messages between parent and child task.

As already mentioned above a collaborative task expression can be interpreted as a bi-parit graph whose nodes are either tasks or temporal operators. Since each task and operator is mapped to a corresponding state chart a bi-parit graph of state charts is created. Edges can be seen as communication channels (similar to CSP (Hoare, 1978)). Thus state charts only communicate with adjacent state charts. This approach supports the concept of separation of concerns and helps to reduce complexity of the communication. Since each collaborative task expression is transformed into a set of communicating state charts and a colabortative task model is defined by a number of task expressions we can accordingly say that the runtime model is defined by a network of sets of communicating state charts.

Up to now we have explained the syntax and rationale of our model. In the next section we elaborate on the modeling of cooperation and dependencies of the environment according to our formally defined model.

3.1 Modeling Collaboration and Comprehensive Dependencies

From the very beginning of classical task modeling the HCI community has regarded objects of the domain as highly related to the task performance. Artifacts and tools are often mandatory to accomplish a task successfully. This fact also applies to smart environments, even though they can be physical objects, devices (stationary or mobile) as well as digital information. Dependencies between tasks and these objects have to be modeled as well to comprise the complexity of the scenario.

Moreover collaboration between actors within the environments has to be supported as well. As pointed out in the scenario there exist various interrelation of task between different users even in simple scenarios. Existing approaches of modeling cooperation of users in HCI lacks flexibility and linkage to objects and devices dependencies. CTML supports both requirements by using an OCL-like language to specify additional execution constraints and effects of tasks.

To execute a task the logical statement in the precondition has to hold which can be either based on the state of the system or the state of an actors' task execution. The abstract syntax of a precondition defined in an EBNF-like notation is as follows:

precondition = *attributePrec* | *taskPrec;*
attributePrec = *identifier DOT check;*
taskPrec = *identifier DOT task DOT state;*
identifier = *(ROLE DOT quantifier)* | *NAME;*

Note that we spare non terminals defined by char sequences (e.g. ROLE; NAME). The first (*attributePrec*) checks whether a set of properties (syntactically defined by the set *F*) has a certain value. Thus, preconditions allow for expressing dependencies of tasks and devices and/or actors. The latter (*taskPrec*) is able to express that a set of arbitrary tasks of actors are in a certain state (E.g. task t1 of actor a1 has to be started before task t2 of actor a2 is able to be started). Note that the life cycle of a task is defined in terms of a state chart whose states can be referenced in preconditions. Additionally preconditions support quantification of actors by means of roles. The meanings of the quantifier are described in Table 1.

Table 1. Semantics of Quantifiers used in Preconditions and Effects

Quantifiers		
All-Quantifier	*allInstances*	All actors of the role have to satisfy the constraint.
Exist- Quantifier	*oneInstance*	At least one actor of the role has to satisfy the constraint.
Non-Quantifier	*noInstance*	The statement holds if no actor of the role satisfies the constraint.

To illustrate the rationale of preconditions some examples for the following precondition are given in Table 2:

(1.) A presenter is allowed to start her presentation after the chairman has announced the talk.

(2.) The listeners are allowed to ask questions after the Dr. Smith has opened the discussion session.
(3.) The chairman can wrap-up the session after all presenters have finished their talk (specified by the property "presented").

Table 2. Examples of Precondition using the Different Features of the Language

#	Role	Task	Precondition
(1.)	Presenter	StartsPresentation	Chairman.oneInstance.AnnouncesTalk.completed
(2.)	Listener	AsksQuestion	DrSmith.OpensDiscussion.completed
(3.)	Chairman	Wraps-UpSession	Presenter.allInstances.presented == true

By the usage of precondition we are able to add execution constraints based on elements of the environment. This comprises the extra complexity of the domain. However to model the dynamics of such a scenario in an adequate manner the effect of a task execution has to be taken into account as well. In contrast to preconditions, effects do not check whether a logical statement holds, but specify the system state after execution the task. Similarly effects either address properties of elements or tasks of actors. The abstract syntax is illustrated here:

effect = attributeEffect | taskEffect;
attributeEffect = identifier DOT assignment;
taskEffect = identifier DOT task DOT message;
identifier = (ROLE DOT quantifier) | NAME

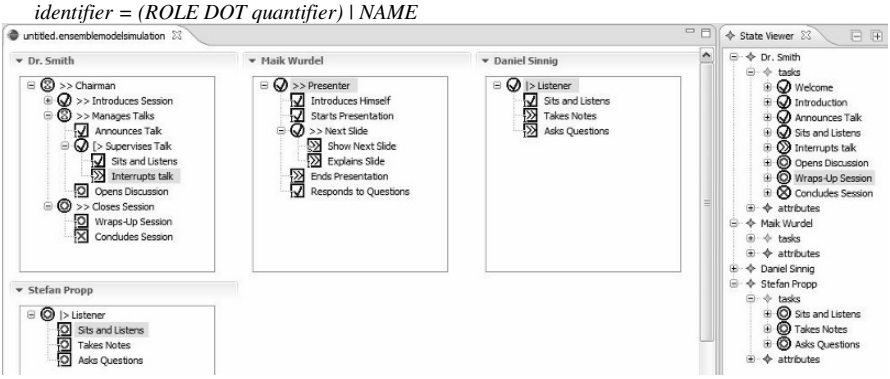

Fig. 2. CTML Simulation incorporating Preconditions and Effects

Please note that an effect on a task is not necessarily taking place since if a message is sent it is interpreted according to the potential transitions defined in the task state chart. For example it is not possible to move from state disabled to running. In this case the running message will be ignored. This avoids inconsistencies and supports the implementation of the message concept. For reasons of brevity we spare examples for effects. The simulation taking into account preconditions and effects to support collaborative task modeling is depicted in Fig. 2.

4. Usability Problems in Smart Environments

Smart environments differ from desktop applications in various aspects, which lead to an according adaptation of usability evaluation methods. Based on the charcteristics of smart environments (Section 1) we derive appropriate evaluation methods. Afterwards we show how to apply these methods for both: providing the users with information about the current state of the system and providing the usability expert with evaluation support.

4.1 Introduction to Usability Evaluation in Smart Environments

The advanced features of smart environments are able to provide a comfortable usage experience, but also introduce new possible usability issues. The reason for usability problems of proactive systems can be decomposed into four potential error components: (1) imprecise sensor values (e.g. wrong location values), (2) misinterpretations of sensor values (e.g. when applying a faulty user movement model to clean the raw sensor data), (3) intention recognition errors (e.g. when predicting the wrong user task) and (4) planning errors (e.g. when delivering the wrong functionality).
To identify these error components we suggest a usability evaluation process comprising three subsequent stages:

(1) Comparing interaction traces (Hilbert, 2000) with a predefined expected behavior to identify possible usability issues.
(2) Analysis of captured sensor data and manual annotations to investigate the reason for the problem.
(3) Investigation of the analysis metrics and visualizations to solve the issue.

Within smart environments a task can be accomplished cooperatively by a number of users by support of their different devices. In addition a certain user can start a task on one device (e.g. a mobile phone with speech input) completing the task later with another device (e.g. a laptop with keyboard). In this case separate interaction traces of the devices can hardly be compared. Therefore we suggest interpreting the interaction trace according to an underlying task model as task trace (Hilbert, 2000). A task trace is understood as arbitrary sequence of performed tasks. Deviations according to the defined temporal order of tasks may occur and need further investigation during evaluation.

Designing a usability test case comprises two activities. First the environment has to be modeled as CTML model and afterwards a usability expert defines the test plan, as it is common practice in usability evaluation.

For the execution of a usability test case we distinguish usability evaluation at different development stages. In early phases, like design, the environment is simulated as an animation of the defined CTML model (see Fig. 2). An interactive

walk through helps to expose weaknesses within the designed artifacts, to revise the underlying CTML models.

After setting up the physical environment, the link to the underlying task models has to be kept to allow evaluation. We provide HTTP access to connect the smart environment and the simulation engine (Fig. 2). During a simulation every leaf task of the simulated task models can be triggered by the events "start" and "stop". These events are internally propagated between adjacent nodes in the task model and cause the task nodes to change the state. All internal and external events are captured to build a task event trace, which is defined as a sequence of events. Each event comprises the corresponding usability test case, the task model, the task, the fired event and a success value. The captured task event trace is used to provide support for both: the usability expert for evaluation (Section 4.2) and the actors within the environment for guidance (Section 4.3). Our approach provides evaluation simultaneously to the test as well as afterwards.

4.2 Visualization and Analysis for the Usability Expert

After capturing a trace of executed tasks and the corresponding sensor data, our approach provides support for identification and analysis of usability issues. To cope with the vast amount of captured data we distinguish between two solutions: on the one hand removing data, which is out of evaluation scope, through filtering and on the other hand keeping all data, but setting focus on data of evaluation interest through aggregation.

For aggregation of the task trace we apply a semantic lens method. Analog to an optical lens a semantic lens is defined by a focus point, a size of the lens and a lens function (Griethe, 2005). Applied to a task trace, the task of interest is focussed, the size of the lens is the number of previous and successive tasks which are covered by the lens and the lens function defines how the aggregation works. The lens function defines the level of aggregation for each position within the lens area (Propp, 2007b). An example for the application of a semantic lens is shown below.

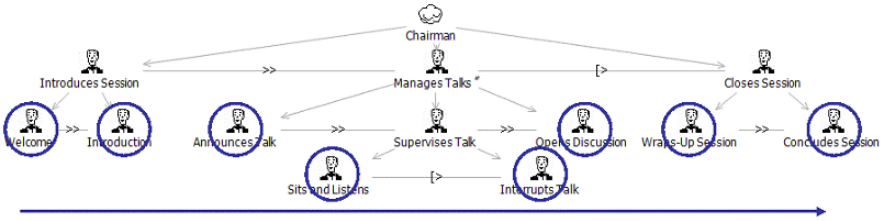

Fig. 3. Complete Example Task Trace

To continue the running example, an interaction trace for *Dr. Smith*, the chairman, is depicted in Fig. 3. The already performed tasks are highlighted. An application of a semantic lens leads to a less detailed trace in Fig. 4. In particular we set

the focus on the task in the center to provide a more concise overview. The more distant the tasks are on the time scale in comparison to the focus, the higher the level of aggregation.

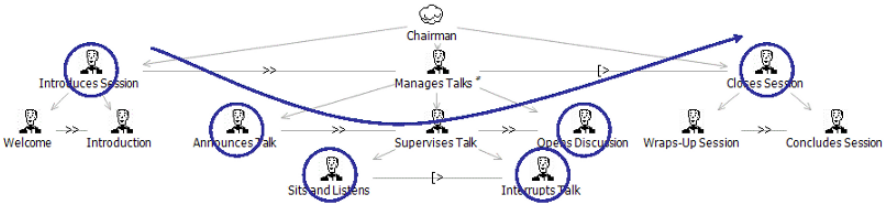

Fig. 4. Aggregated Example Task Trace

The data is captured as a trace with a time stamp for each completed task. The aggregation mechanism analyses the trace to find subsequences which have a common parent within the task tree. Depending on the focus function certain tasks are aggregated and represented by a parent or even more abstract task. The usability expert is able to choose the focus in the time scale and vary the size of focus accordingly. Adjusting the focus function provides a more or less detailed view. The filtered and aggragted task trace can be visualized with different techniques. One simple trace is depicted in Fig. 5.

Fig. 5. Visualization of the Task Trace for a Usability Expert

Our intention is to provide specific visualization techniques for different purposes of evaluation and to have a tool box containing adaptable visualizations. Additonally we provide a timeline view to compare different users according to duration of accomplishing different tasks.

4.3 Visualization for the End User

We intend to support users with a guidance mechanism to visualize the current progress of task execution. Especially in a smart environment it is necessary to provide an overview of the current state of the system. Users might be astonished about some reactions like switching off the light. Hence it might be needed to rollback the system to prior state or forward to a new state.

Therfore we reuse the task trace to provide a history and an outlook of task execution (Propp, 2007a). Traces are prepared in the same way as for the usability expert in Section 4.2. First a filtering stage reduces accomplished tasks, e.g.

- fulfilled within the currently proceeding activity (branch of task model),
- within a predefined time interval in the past,

- with the devices that a user controls.

The subsequent aggregation step applies a semantic lens to provide a more concise overview. Additionally to the performed trace the potential future tasks are derived from the enabled task set, which contains all executable tasks at a certain moment in time. To accomplish the goal the user has a set of possibilities defined in the task model. Therefore the reasonable alternatives are already known and can be visualized as future avenues. We continue the example of the chapter 4.2 and visualize the data for user guindance in Fig. 6.

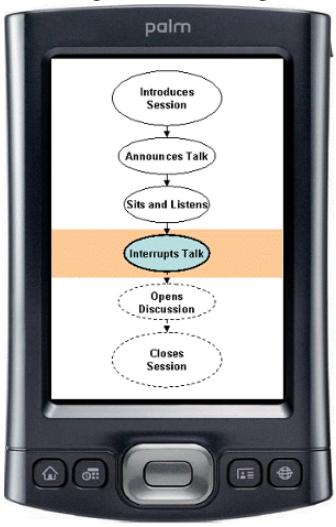

Fig. 6. Visualization of the Task Trace providing User Guidance

The example in Fig. 6 shows the chairmans' PDA to summarize the current situation of the smart environment. The task trace proceeds from top to bottom on a timeline. The focus is automatically set at the currently proceeding task, which is highlighted. The bigger shapes depict tasks at a higher level of abstraction, which are derived within the aggregation stage. The potential future tasks are visualized as dashed oval shapes. Changes within the environment are recognized by sensors and delivered to the usability framework to update the visualization accordingly.

5. Conclusion & Future Work

In this work we presented a collaborative task modeling language which can be used to model the behavior of actors in smart environments. Therefore we extended classical task modeling notation to comprise the raised complexity of the domain such as dynamic collaboration of actors and dependencies of user tasks and the environment. To enable software designers using the notation we devel-

oped an editor as well as a simulator for CTML. In the second part of this paper we elaborated on usability problems within the domain of smart environments. First we highlighted the challenges of usability in this particular domain followed by an approach which makes use of CTML as runtime engine to track the task performance of users. This approach tries to guide the user by visualizing the recent, current and potential future task during task performance. Additionally different visualization techniques are proposed which can help to evaluate task performance by usability experts.

Future research avenues comprise the evaluation of the specification at runtime by a “Wizard of Oz” experiment. This will help us to expose strengths and weaknesses of our approaches based on real data which applies for modeling as well as usability evaluation. Based on these results extending CTML will be another issue of investigation to integrate other elements of the environment. Further aspects of the usability evaluation process will be directly intregated into the modeling environment

6. References

Bomsdorf, B. (2007). "The WebTaskModel Approach to Web Process Modelling." TaMoDia **4849**: 240-253.

Griethe, H., G. Fuchs and H. Schumann (2005). A Classification Scheme for Lens Technique. WSCG (Short Papers) 2005, Plzen, Czech Republic.

Hilbert, D. M. and D. F. Redmiles (2000). "Extracting usability information from user interface events." ACM Comput. Surv. **32**(4): 384-421.

Hoare, C. A. R. (1978). "Communicating sequential processes." Commun. ACM **21**(8): 666-677.

Klug, T. and J. Kangasharju (2005). Executable Task Models. TaMoDia. Gdansk, Poland.

Malý, I. and P. Slavík (2007). Towards Visual Analysis of Usability Test Logs Using Task Models. Task Models and Diagrams for Users Interface Design**:** 24-38.

Mori, G., F. Paternò; and C. Santoro (2002). "CTTE: Support for Developing and Analyzing Task Models for Interactive System Design." IEEE Trans. Softw. Eng. **28**(8): 797-813.

Paterno, F., A. Russino and C. Santoro (2007). Remote Evaluation of Mobile Applications. TaMoDia 2007. Toulouse, France.

Propp, S. and G. Buchholz (2007a). A User Control Mechanism for Smart Appliance Ensembles. KI 2007 Workshop. Osnabrück, Germany.

Propp, S. and G. Buchholz (2007b). Visualization of Task Traces. Interact 2007 Workshop on New Methods in User-Centered System Design. Rio de Janeiro, Brasil.

Shirehjini, A. A. N. (2007). A Multidimensional Classification Model for the Interaction in Reactive Media Rooms. Human-Computer Interaction. HCI Intelligent Multimodal Interaction Environments**:** 431-439.

Sinnig, D., M. Wurdel, P. Forbrig, P. Chalin and F. Khendek (2007). Practical Extensions for Task Models. TaMoDia, Springer. **4849:** 42-55.

van Welie, M., G. van der Veer and A. Eliëns (1998). An Ontology for Task World Models. DSV-IS 98. Abingdon, United Kingdom, Springer.