Task Model-Based Usability Evaluation for Smart Environments

Stefan Propp, Gregor Buchholz, and Peter Forbrig

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

Abstract. Task models are widely used within the research field of HCI for the model-based development of interactive systems. Recently introduced approaches applied task models further to model the cooperative behavior of people using devices within a smart environment. We describe a method of model-based usability evaluation to evaluate interactive systems, with a particular focus on smart environments, which are developed based on task models. We consider the evaluation in early development stages to interactively walk through the models and in later stages to execute a test case within a real environment. The paper provides results of a prototypical implementation.

Keywords: Model-based Usability Evaluation, Task Models, Smart Environment.

1 Introduction

Today, model-based development methods are well-accepted in the field of designing Human Computer Interaction systems. For instance, models are used to describe the user tasks that are to be supported by the system and to specify various environmental aspects, like involved devices and user roles. Based on such models, user interfaces can be developed in a semi-automatic sequence of transformations preserving the models' structure. We are interested in an integration of usability evaluation in all development stages, and will outline (1) the general model-based approach and (2) existing tool support for usability evaluations and then (3) focus on the specific challenges of task-model based development of smart environments and (4) their evaluation.

1.1 Model-Based Software Development

A primary concern of this methodology is that software engineers and user interface designers base their work on the same models: the task model, user model, business-object model and the device model as a representative of a general environment model. These models are as well the basis for the development of the software developed by software engineers as those for the user interface experts. Software development is seen as a sequence of transformations of models that is not performed in a fully automated way but by humans using interactive tools. In [9], the idea of

P. Forbrig and F. Paternò (Eds.): HCSE/TAMODIA 2008, LNCS 5247, pp. 29–40, 2008.

supporting the development of task models by patterns is shown. Task models in this approach are constructed in CTT (Concurrent Task Trees) [6] notation, describing single actions and their hierarchical and temporal relations. One of the tools called DiaTask [8] allows developing a dialog graph that represents the navigation structure of the interactive system. Such a graph is based on the previous specified task model. A dialog graph consists of a set of nodes, which are called views and a set of transitions. There are several types of views to specify different characteristics in terms of visibility, activation and composition. End views are final points in (sub-) dialogs. Each view is characterized by a set of navigational elements. A transition is a directed relation between an element of a view and a view reflecting navigational aspects of user interfaces. DiaTask allows to assign several different dialog graphs to one task model, thus providing support for the development of systems with different user interfaces for different devices and/or groups of users. Given a task model and one or more dialog graphs a simple WIMP style abstract user interface can be generated wherein each task placed on a view is represented by a button on a XUL window. This prototype is refined by a pattern-based replacement of the buttons with more detailed components that help really fulfilling the tasks symbolized by the buttons. The finished UI still keeps the links to the initially task model intact, thus providing an interface to track the users' actions with the system in terms of executing tasks.

1.2 Model-Based Usability Evaluation

There are some approaches to employ task models in usability evaluations. We will have a look at two of them and afterwards introduce the specifics of model based development of smart environment.

The tool RemUsine with its extension MultiDevice [7] uses task models to describe the expected (planned) behavior of users and compares it to the output of another tool component: the tool logger, which is supposed to be available at client-side. The logging tool stores several types of system events during the test session. To provide automatic analysis of the actual user behavior, possible system events have to be mapped to tasks represented by leaf nodes in the task model. This association has to be done once for several user sessions. The tool then provides assistance in analysis by pointing out, at which parts of the tracked user actions the associated task execution violates temporal or logical relations in the task model. The component MobileLogger protocols different types of system and environment variables and includes a dialog based input form for entering these environment conditions. Finally, the tool supports the evaluator in analyzing this potentially huge amount of data by offering different graphical visualizations.

Another tool, developed closely oriented on the model-based development approach outlined above, is the ReModEl ("REmote MODel-based EvaLuation") client-server system [1]. Herein, no mapping of system events to tasks is necessary. One client-side module captures any task-related events within an application developed following the semi-automatic generation and replacement process. These events are sent to the server as they occur and are stored for subsequent requests. An evaluation expert can connect to the server with the same client software but different modules and observe the events related to one or more specific executions. Thus, same-time but different-place evaluations are provided. The client-module shows several

information about the execution, e.g. an animated task tree. There are other modules that allow communication between several clients for example and support subsequent analysis.

1.3 Model-Based Development of Smart Environments

Recent developments aim at transferring the model based development approach into the field of smart environments. We believe that both the actions of users in the smart environment and the functionality of devices within it can be modeled as task models.

The description of each device's capabilities with a task model chunk (device functionality model, DFM) can be found in [12]. The idea is that each time a new device connects to the room's infrastructure its DFM fragment is added to the current task model, referred to as room task model (RTM). Within this process, the combination of available DFMs may provide some new functions, e.g. the conjunction of a scanner and a printer offers a copying functionality. Based on that, the "Task-Constraint Language" (TCL) was introduced in [11]. The actions of every user in the room are described by a task model and constraints specify the modalities of collaboration, e.g. that person "A" finishes his presentation, to give person "B" the floor. Important enhancements have been suggested by Feuerstack et al. [2], extending the task model notation CTT to serve as a runtime model. For that purpose, domain concepts are annotated and an object flow is modeled; different users' task models are synchronized with domain objects.

1.4 Model-Based Usability Evaluation for Smart Environments

Our objective is to provide usability evaluation methods for model based smart environments in all development stages. Because of the models being an inherent part of the system there is no need to parse any log-files in order to extract task-related information. Instead of that we can utilize the task model engine as the source for relevant events. During design time, this engine is used to simulate and animate the underlying models and during run time it acts as the logic within the smart environment providing assistance to the users.

The approach presented in this paper integrates usability evaluation activities in the development process. Furthermore, we believe that software developers and usability experts do not only benefit from working on the same models, but also profit from working in the same environment and interdigitating their work.

Figure 1 shows the process in principle - based on the models (described in section 2) a test case is developed as described in section 3. In section 4 the execution of a test case is explained. Finally, section 5 discusses the analysis of the gathered data.

We define an evaluation scenario as a set of users and devices, each characterized by properties and specific task models. Every user owns one or more roles and all roles are characterized by a certain task model. Every device is associated with one or more types described by a set of properties and a usage model in a CTT like notation, which defines how a device is used. To evaluate a smart environment based on a specific modeling technique the task model chunks to describe user behavior and

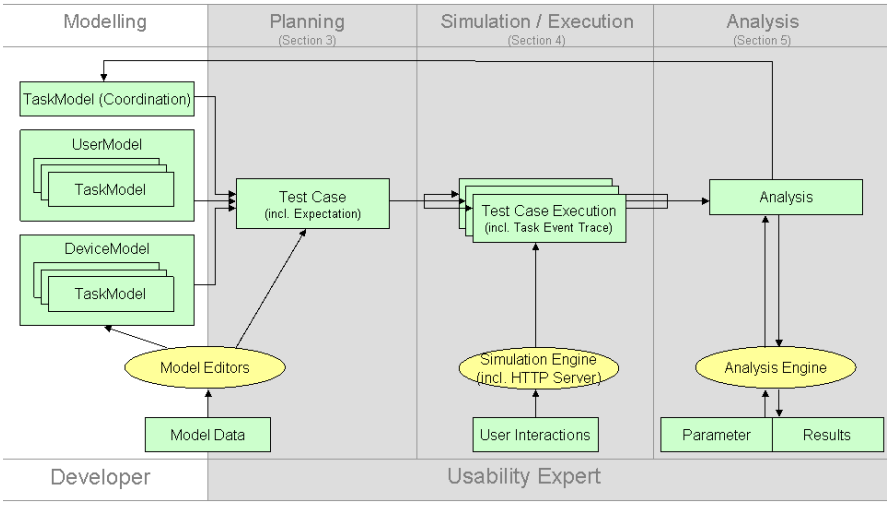

Fig. 1. Usability Evaluation Process

device usage are mapped to CTT notation and additional information is annotated. The aim is to track the interaction between user and environment and validate the interaction according to the model in an analysis stage.

2 Usability within Task Model-Based Software Engineering

In this section we will firstly introduce the vocabulary used in the following. Afterwards, the evaluation approach is presented.

2.1 Disambiguation

According to [6] a task model TM is seen as the sum of possible task traces TT. The hierarchical tree composition from actually executed tasks as leaf nodes and abstract tasks as inner nodes represents the logical structure of the root task that is divided into sub tasks. Within the activities described by a TM there is no contemporaneous execution of any two tasks. At any moment during the execution only one task can be running. This restriction causes some difficulties in describing cooperative work and ongoing activities with several devices in a smart environment. To allow the specification of interactive systems with more than one user acting simultaneously, CTT has been extended to Collaborative ConcurTaskTrees (CCTT) [5]. The main principle in CCTT is to introduce a coordination task tree specifying the relations and interaction between several other task trees that describe the different users or roles involved. In those role-depending task trees a new kind of nodes (connection tasks) is introduced to specify temporal dependencies to connection tasks within another roles' task model. The coordination task model describes these temporal dependencies. In our approach to model the behavior of users and the functionalities of devices we use such separate models for each role and device. Beyond this, Sinnig et al. [11] further

extended CCTT in order to consider the fact that each role is typically fulfilled by different users. Therefore, for each user a copy (instance) of the corresponding role task model is created. The various instances are executed concurrently during runtime and the task constrain language (TCL) specifies synchronization points between several instances.

For evaluation purposes the view on the execution of a specific task is more detailed taking into account the different possible states of a task. States can be enabled, disabled, running, suspended, done, and skipped. Each state change of a task, be it a leaf node or inner node, is called a task event and the execution of a model based system is described as a sequence of such events called Task Event Trace (TET). Events occurring as a result of other events (e.g. the finishing of task "A" enables the execution of task "B") are placed beyond the causing event in the sequence.

During the preparation of a usability evaluation of a smart environment system an expected behavior is specified called Expected Task Event Trace (ETET). This ETET is used during the evaluation and the afterwards analysis to compare the actual behavior of users and devices with the expected.

2.2 Brief Overview

This section describes the evaluation process in principle, details of evaluation techniques and visualizations can be found in the subsequent sections.

For smart environments in the development stage we suggest a test setup wherein at least two experimenters act as mediators between the smart environment room and the task model interpreter. Such "Wizard of Oz"- experiments are a common technique for early stage tests of window based software systems and have been conducted within a large number of projects.

The goals of the evaluation can be divided into two sub goals: One aim is to validate the task models (models for roles, devices and the coordination task tree) and the other one is to identify weak points in the environment's sensors and the interpretation of the users' behavior. We will outline the procedure of a usability evaluation and point out, which kinds of problems are addressed.

The evaluation's preparation starts with the definition of a scenario that is to be carried out by one or more users in the smart environment room. Therefore, the task models of all devices and roles participating in the proposed scenario are gathered and an expected behavior has to be developed. This can be done by recording a usage session conducted by experts or by defining a trace manually.

The users taking part in the evaluation are now instructed to fulfill the tasks defined in the scenario. They do not know the complete task models in detail but only the goal and a list of sub goals so as to avoid them to behave more unnatural than inevitable.

During the evaluation the experimenters are provided with all information that is produced by the sensors in the smart environment room, video streams from cameras placed in the SE and an audio stream to keep track of the users' activities. Furthermore, the current states and properties of the devices are displayed. All these data flows are recorded to be used in subsequent analysis, too. We developed an Eclipse plug-in to simulate multiple task trees describing a role's action or a device's capabilities and functions. An experimenter can define a set of task trees for an evaluation and

activate the simulations simultaneously. Each time the experimenter observes an action he journalizes it using the task model simulation.

Another expert is responsible for initiating the effects on devices in the SE. Tasks marked as system tasks in the device models are started and stopped in the simulation and delivered as commands to the appropriate devices. Thus, the information flow out of and into the smart environment room is complete: Observed actions are recorded using the manual triggered model simulation and an additional protocol for tasks not modeled so far, and the environment's reactions are simulated by an operator sending the commands to the devices in the smart environment room. The users performing activities in the smart environment room interact with the room devices as if the task model engine was triggering them according to the task models. The subsequent analysis of the recorded events reveals several shortcomings of the SE system developed so far.

The next sections show how to prepare and conduct such an evaluation using the tools that are developed.

3 Planning a Usability Evaluation

Within Eclipse all artefacts and documents necessary to conduct a usability evaluation are defined in a UsabilityTestCase file. This includes a description of the evaluation preparation and environment as well as a definition of the test case that is to be executed. According to [9] the test plan specifies the required resources, focuses the points to test, and serves as a communication tool between the different members of the working team. It includes a high level description of the test's purpose, a list of all the questions and objectives that are to be solved by the test as precise as possible, a description (profile) of all users involved, a detailed description of the test execution, a list of tasks to be carried out by the users in an appropriate level of abstraction, and a description of the used equipment and of the level of participation or neutrality of the test conductors. In the context of model-based smart environments also the task models of both user roles and device types are included, too. The definition of the test executions can include instructions for the usability experts to ask questions to the users during and/or after the test execution. These questions are linked to tasks in the according task model and the links can contain filter criterions to define several questions as to be asked under specific circumstances only.

All information about one single execution of such a UsabilityTestCase is gathered in a UsabilityTestCaseExecution file. Here, the recorded task event traces and the description of the actual setup are stored. It also contains comments and notes made by the usability experts during the evaluation. Figure 2 shows some elements in a usability evaluation: The task models of users and devices as well as some artefacts that are involved in the test case.

4 Conducting a Usability Evaluation (Simulation/Execution)

Now a usability test case is defined, which comprises a description of the test plan, the physical environment (e.g. needed device types) and involved user roles. This

abstractly defined test case can be carried out several times with different concrete users and concrete devices, which fit into the defined user roles and device types. Conducting the same test case several times with slightly different parameters ensures the statistical significance of the test result. It further allows identifying infrequently occurring usability issues and comparing the behavior of different users.

4.1 Capturing a Task Event Trace

During the usability evaluation the users are observed and the users' interactions are captured. Capturing can be accomplished at different levels of abstraction, beginning with low level events at the physical level (e.g. mouse clicks and a person changing the location), up to the problem-oriented level (e.g. a person gives a presentation) [3]. We capture the interactions corresponding to an underlying task model, which describes for instance the behavior of a certain user role. Each record of the captured task event trace comprises a time stamp, the conducted test case, the task model, the task, the fired event and the success value. The applied task model simulation engine [8] instantiates a task model and conducts a simulation through receiving events from user interactions and exchanging events between task nodes within the task model. Events cause state changes of the task nodes. For instance a task with the state "enabled" is caused by the event "start" to change the state to "running" and a subsequent event "stop" leads to the state "finished". Depending on the current state of a task, some events are prohibited. For instance a task with the state "enabled" can not react on a "stop" event, because the task has to be started first. In these cases no state change occurs, but nevertheless the event is captured and marked as not successfully processed. Beyond the events which are directly influenced by the user interactions and sent to the leaves of the task tree, there are a lot of events exchanged between inner nodes of the task tree. A task which changes the state sends a notification event to the parent node which updates the own state and sends notifications to the other children and the own parent. We have enhanced the task model simulation engine to capture all these events as a task event trace.

4.2 Simulating and Executing an Evaluation

The task model simulation engine can be used at different stages of the development of the smart environment. In early stages the task models can be simulated to evaluate the content and structure of the task model. In later development stages the task model based developed smart environment can be evaluated. We provide tool support to simulate the execution of a smart environment. The figure below depicts the simulation UI. The left hand side provides the design time view at the usability test case. The right hand side provides the runtime view at the test execution. Every element of the test case on the left is simulated as a task model at the right. A graphical symbol in front of every task within the task tree reflects the current state of the task instance. For instance a red cross marks a task as "disabled", a green circle as "enabled" and a blue tick mark as "finished". The user triggers the simulation progress via context menu. Each leaf task provides a menu to fire the events "run", "start", "stop" and "crash", depending on the current state of a task. "run" is a composition of "start" and "stop". "crash" aborts the execution of a certain task. The different task models are simulated concurrently.

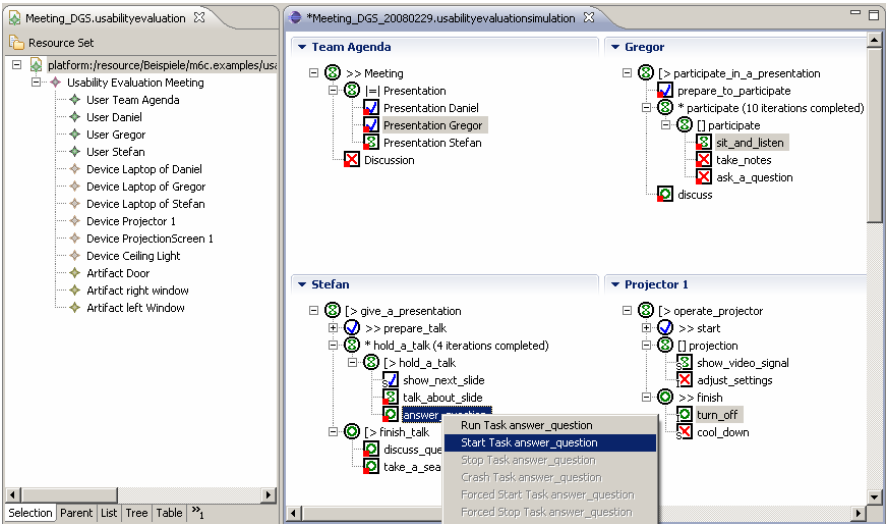

Fig. 2. Usability Evaluation UI to execute a Test Case

To evaluate a real smart environment, the task engine can also be applied. The behavior of the users within the environment, like moving to the presentation zone or switching a projector on, has to be recognized and a corresponding event has to be sent to the task model simulation engine. We provide two complementary ways: on the one hand a HTTP connection to connect external devices to the simulation engine and on the other hand a "Wizard of Oz"-based technique with a human operator.

(1) We enhanced the simulation engine with an HTTP server to receive external events from various sources like a smart environment. A HTTP request from the smart environment contains the destined task and the event. The task is defined by the test case, the task model, and the contained task. A smart environment is equipped with sensors to detect the user behavior and to provide user support accordingly. Therefore the fulfilled tasks are recognized within the normal execution of the environment and subsequently used to trigger the task models executed within the simulation engine. In the case that the smart environment deploys a different modelling technique, the tasks have to be elicted and mapped to models for the simulation engine. Events can only be sent to leaf tasks. The events "start", "stop" and "crash" are supported. A test case execution registers at the server to receive events. For every incoming event, the corresponding test case is determined and the destined task model and task instance. After sending the event the task model engine reacts accordingly and changes the states of the destined task and the dependent tasks. The task event trace is enhanced. If an event is currently prohibited, the event is captured as failed within the event trace. The HTTP response contains optionally a success notification, the resulting set of enabled tasks or the caused events as captured in the task event trace. During the test execution the evaluation UI (figure 2) is concurrently updated to visualize the current state of the environment.

(2) A "Wizard of Oz"-experiment can be conducted alternatively for smart environments which are not directly connected to the simulation engine and therefore are

not able to deliver the recognized user interactions. An additional person observes the interactions within the environment. This observer transfers the user behavior into the simulation engine. The user interface depicted in figure 2 allows to manually trigger the simulation engine. The advantage of this manual process is that an observing person perceives even unexpected situations and copes with them. For example an unforeseen task which was fulfilled by a user can be captured for later analysis. But in a more complex environment the observer might be overstrained by the vast amount of events, which have to be manually processed. Hence we recommend combining both evaluation techniques. Most tasks are fulfilled according to the defined task model and therefore can be captured with the HTTP connection between smart environment and evaluation engine. Some tasks are unforeseeable and can only be captured by a human observer.

When executing a usability evaluation we basically distinguish between three types of tasks according to the task model:

(a) A user fulfills a task which is currently allowed in the task model. The observer triggers the task with the appropriate event in the simulation engine.
(b) A user fulfills a task which is contained within the task model, but currently prohibited due to temporal relationships. In the evaluation UI we provide the option "forced start" to capture the start of a task immediately. The evaluation engine captures the event as failed, but doesn't affect the task model state. A "forced stop" ends a task in an analog way.
(c) A user fulfills a task which is not defined within the task model. The evaluation UI provides the observer with the option to enter an additional task, which is captured as failed event within the task event trace analog to (b).

The evaluation UI depicted in figure 2 can be deployed in different situations. In early development stages it is used to simulate a walk through the task models and to check their consistency. After the smart environment is set up, the HTTP connection to a real environment triggers the evaluation UI and additional manual events can be sent.

5 Analyzing the Results of a Usability Evaluation

During the execution of a test case interactions are captured as task event trace. This data can be analyzed at the same time or later after conducting a test. Conducting the analysis in parallel to the test has the advantage that issues are recognized immediately offering the opportunity of giving support to test users. For instance if an issue is already known from another test, but currently not fixed, it might be beneficial to provide the users with help to save time and discover further issues.

An analysis is based on a single test case and an arbitrarily chosen number of test case executions of the same test case. Hence an analysis can examine both a single test execution in detail and several different executions in comparison to each other. Comparing several executions allows varying a specific parameter for detailed evaluation while preserving the other parameters. Examples are the evaluation of differences of specific user groups (e.g. novice vs. advanced users), under certain context influences (e.g. light intensity, furniture arrangement) or alternative implementations of the smart environment. The task models describe the behavior of people

and the potential usage of the devices on an abstract level, leaving out the implementation details. Different implementations can satisfy the specified task models and a comparison identifies individual strengths and weaknesses. We suggest employing some expert users to carry out the test case first and compare their task performances to other users'.

Figure 3 continues the example from section 4 and depicts the analysis in parallel to the evaluation. The left hand side contains the evaluation of a test case, while the right hand side shows a visualization of the current evaluation progress.

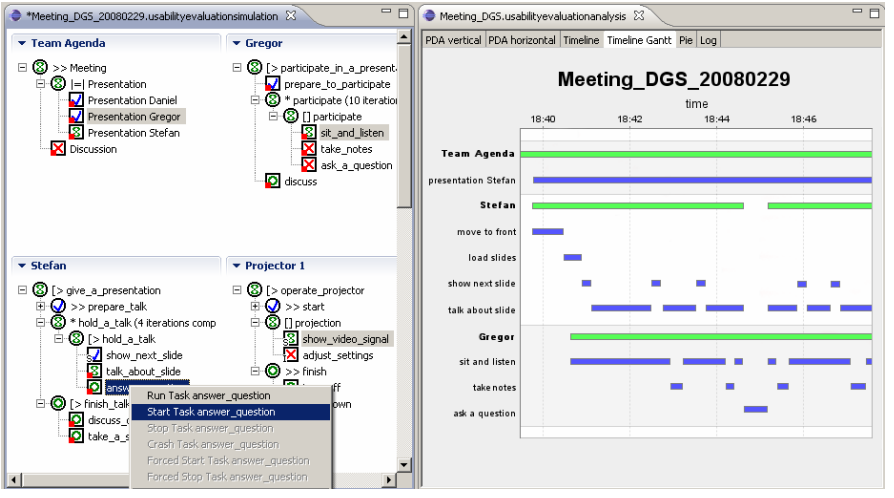

Fig. 3. Analysis of a Usability Test Case

The gantt chart depicts the interactions of the ongoing meeting according to a timeline. The completed tasks are grouped by their task model. A task model is printed in bold case and the duration of executions of the containing tasks is depicted as a green bar. The already started leaf tasks of the task model are depicted as blue bars to indicate their duration. The different task models are executed concurrently.

The captured task event trace serves as basis to derive the input data for the analysis and visualization. The leaf tasks executed by users are extracted and the successfully as well as the failed events are taken into account. Failed event executions reflect a conflict between user behavior and task model, because a user in a smart environment can always choose freely which task to execute. For instance the user can suddenly skip the presentation because of a headache even if the task model doesn't consider this exception. Therefore the evaluation UI allows the observing expert to capture this unexpected task, while the event execution is marked as failed according to the specified task model. Unexpected tasks are depicted as a red bar. A bar depicts the duration of the task execution, while a task is in the state "running". Therefore the events "start", "stop", "crash", "suspend" and "resume" are analyzed.

Depending on the individual interest of the usability evaluation the captured data can be filtered and aggregated. Filter options comprise the filtering according to the minimum and maximum duration of tasks, the start and end time, the involved users and devices. Furthermore data is prepared by aggregation. If there is a huge amount of executed tasks, which are very short and represent a high level of detail, a sequence of leaf tasks with the same parent node within the task model can be aggregated to a higher abstraction level.

An additional normalization step is applied when comparing multiple event traces of different evaluation executions. When different persons fulfill the same task and differ in the needed time, this might be an indicator for a usability issue. But in some cases we faced the problem that the normal working speed was greatly varying and there was no system related issue, e.g. if one tester needs significantly more time for changing slides forth and back with the presenting device. We overcome this problem by normalization of the durations of the task sequence, which proceeds as follows: the expert chooses a task and a certain user. The according time captured during observation is compared to the other users' and for each user is a factor derived. The factor is applied to all tasks of the respective user. As a result the duration of slower users is stretched and the duration of faster users is compressed. We suggest designing a short calibration task and appending it in front of the evaluation. As an alternative all overall durations can be set to the same duration, to compare all users at a 100% basis.

Furthermore timelines are provided to compare different test case executions at a glance. Parallel lanes depict different executions of the same test case with different persons and in addition to [4] the normalization can be applied.

6 Conclusion

In this paper we presented a usability evaluation approach for the task model-based evaluation of interactive systems, particularly for smart environments. Within a smart environment a task can be accomplished cooperatively by a number of users, while changing the devices during executing a task. For instance a user can begin a task on a mobile phone with speech input and fulfill the task on a laptop with keyboard. An interaction trace based only on physical events alone is in this case not sufficient, because it is difficult to compare voice and keyboard input. Therefore we enhanced the physical interaction trace with device independent information on the task level. We defined usability test cases based on task models, to capture task event traces. A subsequent analysis and visualization allows the identification of usability issues. For interactive systems, which are designed with task models, the identified issues within the task models are directly related to the design model. Hence the problem can be tracked back to the cause within the development stage.

To provide tool support we enhanced an existing task model framework with usability evaluation facilities. Hence the evaluation is directly integrated into the model-based development process and allows rapid usability testing at different development stages.

Future research avenues comprise a case study to evaluate our technique within our smart environment. This will help us to discover strengths and weaknesses based on real data.

Acknowledgments. The work of the first author was supported by a grant of the German National Research Foundation (DFG), Graduate School 1424.

References

1. Buchholz, G., Engel, J., Märtin, C., Propp, S.: Model-Based Usability Evaluation - Evaluation of Tool Support. In: HCII 2007, pp. 1043–1052 (2007)
2. Feuerstack, S., Blumendorf, M., Albayrak, S.: Prototyping of Multimodal Interactions for Smart Environments based on Task Models. In: AMI 2007 Workshop on Model-Driven Software Engineering, Darmstadt, Germany (2007)
3. Hilbert, D., Redmiles, D.: Extracting Usability Information from User Interface Events. ACM Computing Surveys 32(4), 384–421 (2000)
4. Malý, I., Slavík, P.: Towards Visual Analysis of Usability Test Logs. In: Coninx, K., Luyten, K., Schneider, K.A. (eds.) TAMODIA 2006. LNCS, vol. 4385, pp. 25–32. Springer, Heidelberg (2007)
5. Mori, G., Paternó, F., Santoro, C.: CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. IEEE Trans. Softw. Eng. 28, 797–813 (2002)
6. Paternò, F.: Model-Based Design and Evaluation of interactive applications. Springer, Heidelberg (1999)
7. Paternò, F., Russino, A., Santoro, C.: Remote evaluation of Mobile Applications. In: Winckler, M., Johnson, H., Palanque, P. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 155–168. Springer, Heidelberg (2007)
8. Reichart, D., Forbrig, P., Dittmar, A.: Task Models as Basis for Requirements Engineering and Software Execution. In: Proc. of. TAMODIA, Prague, pp. 51–58 (2004) ISBN 1-59593-000-0
9. Rubin, J.: Handbook of usability testing. In: Hudson, T. (ed.) Wiley technical communication library (1994)
10. Sinnig, D., Gaffar, A., Reichart, D., Forbrig, P., Seffah, A.: Patterns in Model-Based Engineering. In: Proceedings of CADUI 2004, Madeira (2004)
11. Sinnig, D., Wurdel, M., Forbrig, P., Chalin, P., Khendek, F.: Practical Extensions for Task Models. In: Winckler, M., Johnson, H., Palanque, P. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 42–55. Springer, Heidelberg (2007)
12. Trapp, M., Schmettow, M.: Consistency in use through Model based User Interface Development. In: Trapp, M., Schmettow, M. (eds.) Workshop at CHI 2006, Montreal, Canada (2006)