Defining Expected Behavior for Usability Testing

Stefan Propp and Peter Forbrig

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

Abstract. Within HCI task models are widely used for development and evaluation of interactive systems. Current evaluation approaches provide support for capturing performed tasks and for analyzing them in comparison to a usability experts' captured behavior. Analyzing the amount of data works fine for the evaluation of smaller systems, but becomes cumbersome and time-consuming for larger systems. Our developed method aims at making the implicitly existing expectations of a usability expert explicit to pave the way for automatically identifying candidates for usability issues. We have enhanced a CTT-like task modeling notation with a language to express expected behavior of test users. We present tool support to graphically compose expectations and to integrate them into the usability evaluation process.

Keywords: Usability Evaluation, Task Models.

1 Introduction

Task models are widely used within the domain of Human Computer Interaction. For eliciting requirements task models describe the progress of task execution to accomplish a certain goal. Subsequent development stages apply task models as initial artifacts for model-based development of user interfaces [9]. Several approaches further exploit task models for usability evaluation. Examples are RemUSINE [6], ReModEl [1] and the task-based timeline visualization [4]. Those evaluation techniques capture the observed user interactions on a lower level of abstraction (e.g. mouse clicks or sensor values of user movement), which can be easily captured but the vast amount of data is difficult to interpret. Subsquently the sequence of captured events is lifted from an interaction level (e.g. button click) to a task-based abstraction (e.g. printing a document) [8], which allows interpreting the results in a more natural way. Hence a usability expert can conveniently compare the observed behavior of test users with his/her expectation of efficient task performance to reach the goal of the test case. Deviations indicate candidates for usability issues. This comparison is carried out as a manual process with tool support for visualizing a task trace, but lacking an integration of machine-readable expectations.

The approach in [6] goes a bit into this direction. It offers a comparison between two task traces: the observed trace and an "ideal path". A designer specifies this path and the degree of deviation can be visualized. However, there is no opportunity discussed to generalize the expectation to cover different expected traces. For instance

M. Kurosu (Ed.): Human Centered Design, HCII 2009, LNCS 5619, pp. 110–119, 2009.

the task "sending an email" can be accomplished in different ways. It is appropriate either to use a web interface or to start your email client. Both ways solve the task. The IBOT system [12] also provides a mechanism to capture user interactions and further automatically compares the behavior of user and designer. WebQuilt [2] visualizes the navigation path through a website. It visualizes the observed path of users in contrast to a designers' expected path, which is a comparison between two navigational pathes.

As summarized there are some approaches available for automatically comparing two captured logs with each other, which may be actually logged or even designed, but do not allow to specify some degree of freedom in a sense that a user might deviate from the expected behavior in certain aspects. Therefore existing methods work fine for the evaluation of smaller systems, but become cumbersome and time-consuming for larger systems.

We aim at overcoming this problem by defining expected user behavior in a machine-readable form with some degree of freedom for deviations. Therefore we have enhanced our CTT-like task modeling notation with a language to specify expectations. In the general case when we evaluate an artifact without an existing relationship to a task model, such a task model has to be modeled and extended to form the expectation. In certain cases when we evaluate a piece of software which was developed task-based we reuse these models and enhance them.

In section 2 we draw the bigger picture and discuss the integration of our concept into the usability evaluation process. Section 3 explains the method and provided tool support, exemplified with modeling expectations for people interacting with each other in a meeting situation. We show how to specify expectations, capture according user behavior and finally analyze the results. Section 4 gives a conclusion and future research avenues.

2 Usability Evaluation Process

Before we go into the details of specifying expectations, we give an overview of the whole usability evaluation process, where our approach fits in.

The process comprises four stages (see figure 1): modelling, test planning, test execution and analysis of test results [7].

1. The modelling stage should be carried out during product development. During requirements analysis tasks are elicted, which should be carried out by users. These tasks are put into relationship to each other. A designed task model in CTT notation can be built [5], which reflects a hierarchical decomposition of tasks into subtasks. These tasks are interconnected with temporal relations. Each task model describes how a user can achieve a certain goal. Furthermore a user may have different roles. For instance a meeting participant can switch between the roles "presenter" and "participant". Each role's available tasks are described by a respective task model. Additionally a task model for coordinating the other task models can be provided [5].

2. In the planning stage a test case is defined as it is common practice for usability evaluations. We specify for instance purpose, test objectives, a description of the environment and the evaluation measures [10]. We enhance this textual information with task models describing the possible user behavior while using the evaluated

artifact. When specifying a test case, the usability expert already has an expectation in mind, how to perform the tasks in an efficient way. This implicit knowledge should be made explicit as machine-readable expectation model.

3. In the execution stage a test case is conducted several times with different test users. Observations (like key strokes, mouse clicks, location coordinates of moving users and video streams) are captured and annotated by an expert. Our test environment provides an evaluation engine, to collect this data from the physical environment with sensors and video cameras. During evaluation the captured user behavior is compared against the expectation to discover deviations. These evaluation results are additionally captured.

4. In the analysis stage an analysis engine provides capabilities to analyze and visualize captured data. Finding usability issues within the vast amount of data is a tedious task, because often it is not intuitively clear how an issue may look like. To cope with that, we particularly emphasize on expectations, because deviations from the expected behavior can be derived automatically, through comparing expectation and accomplished task trace. The result set contains some candidates for usability issues, leading to a reduction of relevant data. Subsequently a usability expert can focus on examining data interrelated with identified situations. Interactively walking through video streams, annotations, sensor and task data, helps identifying causes of an issue and improving the underlying task models to better describe user behavior.

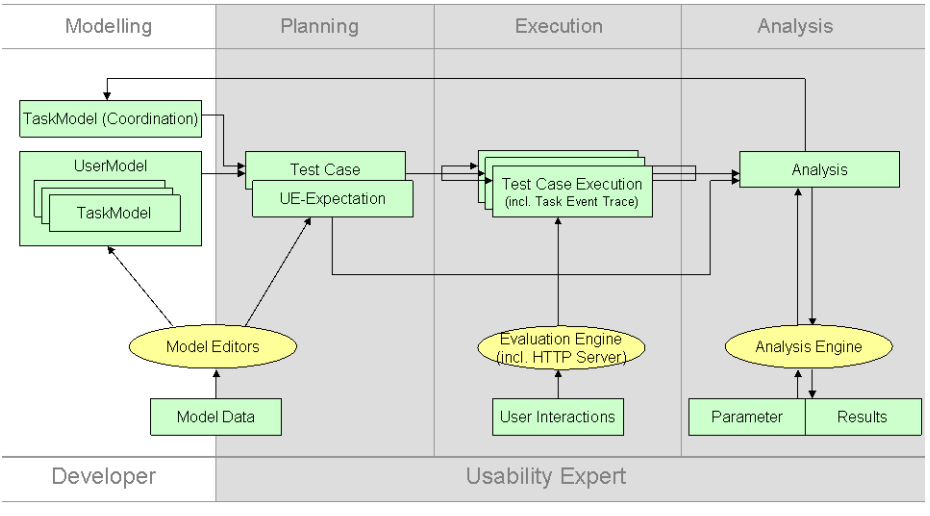

Fig. 1. Task Model-based Usability Evaluation Process

3 Specifying Expectations

3.1 Example

First we introduce a running example, which is subsequently used to describe how to specify expectations before a test starts, how to evaluate expectations during the test

and how to analyze the results afterwards. Our approach aims at evaluating a wide variety of artifacts, including software and physical artifacts, where a task model can describe the interaction of a user.

According to our prototypical implementation, we consider a meeting situation within a collaborative environment. Initially no persons are present. The room is only equipped with furniture and some stationary devices, like projectors at the ceiling and movable window blinds. Before the meeting begins, three people A, B and C are populating the room, while carrying their personal devices with them. Their PDAs and laptops contain slides for the presentations and help with taking notes during listening to the other talks. All three people have to give a presentation in an arbitrary order, closing with a discussion. Finally they exit the room, carrying their devices. The room senses the loctation of the people and notices if someone takes up a device or another item in the room, like a laser pointer. The environment tries to derive which task is carried out next. For instance if a person moves to the front, connecting the laptop with projector, while the others are sitting, the environment, derives the beginning of a presentation and gives support. It shuts the window blinds at the front and moves down the appropriate projection screen. The evaluation should discover strength and weaknesses within the interactions between meeting participants and the surrounding environment. Particular usability questions to investigate within this domain are: Does the environment derive the correct user behavior from the sensor data? Are the users' performed tasks appropriately supported by the pro-active meeting assistance?

3.2 Method

To evaluate the usability of an artifact we begin with a task model which describes how users can interact with the given artifact. If we concider a software artifact, this task model may already exist from requirements elictation or task model-based development [9]. In other cases it has to be modeled first. A task model describes a set of sequences of performed tasks to reach a goal. In most cases several alternative task traces reach the goal.

A usability test typically focusses on certain functionalities of an artifact, especially when the artifact is still under development and some parts are not implemented yet. Therefore users carrying out a test case are expected to perform only tasks contained within this corresponding subset of the task model. Other tasks are possible but out of scope of the current test case, since they do not support reaching the given goal.

We distinguish between a task model describing a bunch of functionality offered by the artifact on the one hand and an expectation as subset focussing at the tested functionality. The expected task performance is further constrained to devices which have to be used, certain context conditions and maximal durations for task performance. An expectation is further described as follows:

A CTT model describes a set of possible task traces to interact with the artifact. The expectation model is build on top of this task model and comprises additional annotations to constrain these traces. To evaluate an artifact under different test conditions, for each test case a separate expectation is defined. All expectations may

constrain the same model in another way depending on the designers' expected user behavior.

An expectation consists of a set of expectation statements and can be described in an EBNF-like notation:

```
expectation    = task ":" (event {"," event}) ":"
                 {statement ";"};
event          = START | END | ENABLE | DISABLE |
                 SUSPENT | RESUME | ABORT | SKIP;
statement      = classification ":" expression;
classification = PERFECT | GOOD | BAD;
```

An **expectation** is specified for a certain **task**. The example (table 1) contains several task models, one for each user role. Therefore the task has to be qualified with the respective task model as "participant.present". To evaluate expectations a task model engine was incorporated [9]. During carrying out a test case each task within the task model has a state, for instance a task begins typically as "enabled", turns into "running" and finally into "finished". State changes are triggered by events. A user can only perform leave tasks of a task model and therefore cause the task engine to fire "start" or "end" at these tasks. Each expectation is evaluated when the specified **event** is fired. Several **statements** can be associated, which are evaluated sequentially. The contained **expression** is an OCL-like expression to navigate within the task model and evaluate the tasks' attributes. Accessable attributes during runtime are for instance the states of tasks, applied devices, other involved users, the needed duration for task performance and context information. Context information depends on the available sensors. In our test environment we use mainly location sensors and RFID sensors to capture involved devices. Further context information can be annotated manually or provided via additional sensors. We use OCL [3] to specify these expressions. OCL is very expressive, while some expressions are long and difficult to read. Therefore we provide some helping functions for a more convenient navigation for the domain of task modeling, like it is discussed in [11]. The evaluation of such an expression results in a boolean value, which is interpreted in OCL as a constraint which is satified or not. We prefer a more fine grained grading. Therefore we classify a user interaction according to the degree of desirability within the current situation. We distinguish the **classification** as "perfect", "good" or "bad". For instance to perform the task "give a presentation" (a) it is goal-oriented to "load slides" (hence classified as "perfect"), (b) it is destructive to "leave room" (hence classified as "bad") and (c) optional to "open a window" to get some fresh air (hence classified as "good"). Currently we work with these three categories, but it is also possible to distinguish more or less categories. The list of **statements** is sequentially evaluated. Each OCL expression is handed over to the parser. In case of a result "true" the associated classification label is returned; in case of "false" the next statement is evaluated. Hence the first match of a statement determines the result. The statement at the end of table 1 "bad : true;" ensures the result "bad" if nothing previously matched.

Table 1. Examples for Expectation Statements

Task	Event	Classification	Expression
participant.present	start	perfect	self.device.includes (presenter_device)
		perfect	self.context.includes (presentation_zone)
		good	true
participant.present	end	perfect	self.duration() < 300
		good	self.duration() < 600
		bad	true

Table 1 exemplifies the method with two simple expectations, each comprising three statements. When a person starts to "present" the used device and location within the room is evaluated. When finishing to "present" the time of the presentation is measured. Within a test case the persons are asked to present 5 minutes (300 seconds). If the persons within the room face serious issues, preventing them from directly performing the tasks described in the test case description, often the duration needed expands. In this example a duration of more than 10 minutes is defined as threshold to mark the "present" task as potential usability issue, which has to be further investigated .based on archived video and sensor material of that test session.

3.3 Graphical Tool Support

Composing statements on a textual level allows a very accurate specification of expectations. But beyond the very simple example in table 1 real world examples are much more complex and from our experience it is a very tedious task, because composing statemens manually is monotonous and error-prone due to a high degree on redundancies. Hence we provide a GUI to graphically compose expectations (figure 2) and automatically generate the according expectation statements.

On the left-hand side a tree view visualizes the task model, while the right-hand side depicts a gantt view of the timeline. The navigation within the task tree allows expanding and collapsing the task lanes at the right. Time constraints are set via drag and drop in the gantt view. The colors green and yellow mark the maximum length of a task, while red marks a task which should not be performed within the current test case. Arrows mark additional temporal dependencies. For instance if the task model allows the presentations of persons A, B and C orderindependently, an arrow from "A to B" in the expectations requires the presentation of A to be finished before B starts. If a task is not depicted as gantt lane, there is no expectation set. Normally only a few activities of interest are specified. When selecting a task details are displayed in a properties view at the bottom of the screen to adjust further parameters. Necessary devices, other involved users and certain context conditions are specified. Context parameters can be customized as necessary for the individual evaluation. Examples are the location of users within a room, touched items, light conditions, medical parameters of testers, manually annotated mental workload or categories of emotions. Arbitrary annotations are possible. Parameters can be specified for each occurrence of the task separately or globally for each repetition of a task.

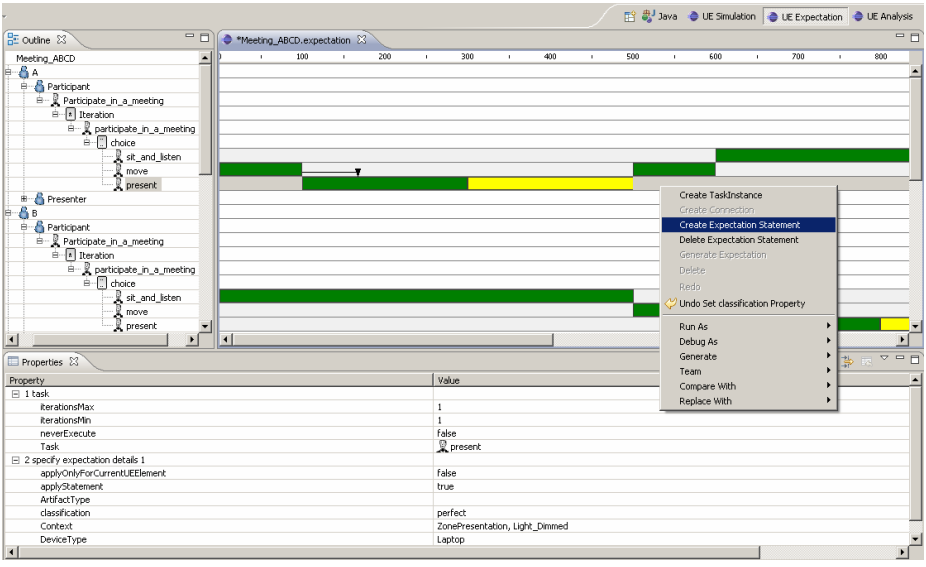

Fig. 2. Specifying Expectations

Our first approach was starting from scratch with a white gantt view, allowing a usability expert to draw task lanes and type in context parameters. To save some time we offer the possibility to load a captured test session which is close to the expected interactions. The data only needs to be adapted in certain aspects, for instance adjusting durations and deleting some unrelevant tasks while adding some missing information.

After having finished the graphical specification, expectation statements are automatically generated. For different modeled examples the expressiveness was adequate. If a usability expectation is to be defined which exceeds the opportunities of the graphical notation, the generated statements can be manually refinded to include arbitrary OCL expressions.

3.4 Test Case Execution

To test the evaluation approach we have developed an evaluation application (figure 3). Following the running example we focus on evaluating meeting situations. The lower left part depicts a bird's view of a room with some grey tables, grey chairs and the participating persons. The upper left part contains the animated task models for the three persons showing the current progress of task performance. Via drag and drop persons are moved through the room while task performance can be triggered within the animated task models. Further annotations are possible. While interactively walking through the specified environent's models the expectation are evaluated.

To replay real world data, at the upper right side a recorded test session from the physical environment can be loaded. Movements within the room and performed tasks are visualized accordingly at the left-hand side.

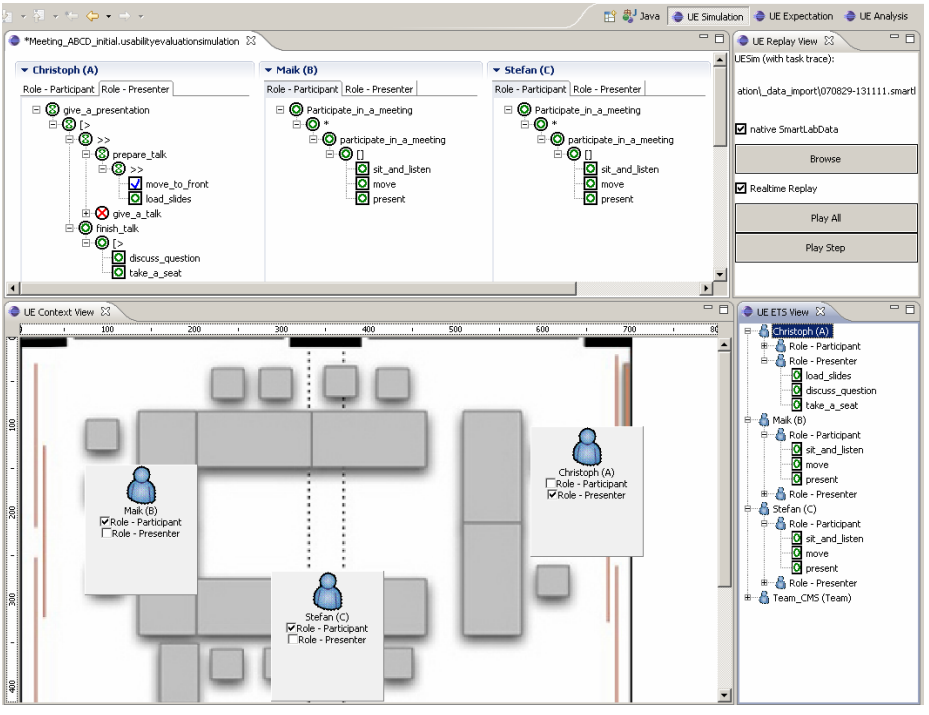

Fig. 3. Data Capturing

3.5 Analysis

After the testers have performed all tasks of the test case, the captured data has to be analyzed. To cope with the vast amount of data, like sensor data, video streams or annotations, we focus on results of the expectation evaluation. Figure 4 depicts the current state of an ongoing implementation. The upper part depicts the actually fulfilled tasks of the testers as gantt timeline according to a task model at the right. The views in the center allow interactively exploring captured data. Filtering options for instance comprises the filtering for certain users, for tasks with very short or very long durations and specific expectation results. Filtering for tasks which were performed "bad" lists situations with major deviations from the expectected behavior, which indicates candidates for usability issues. A subsequent investigation of video streams and sensor data examines whether it is a real issue and to identify the cause. We try to avoid that all captured data has to be examined again. Instead an expert can focus on the automatically discovered issue candidates.

The suggested analysis has also some limitations. A prerequisite is a well defined expectation. Otherwise real issues might be erroneously overlooked. The automatical identification should only be a first step in analyzing evaluation results. A careful investigation of captured data and even uncaptured details visible at the videos should complement the presented approach.

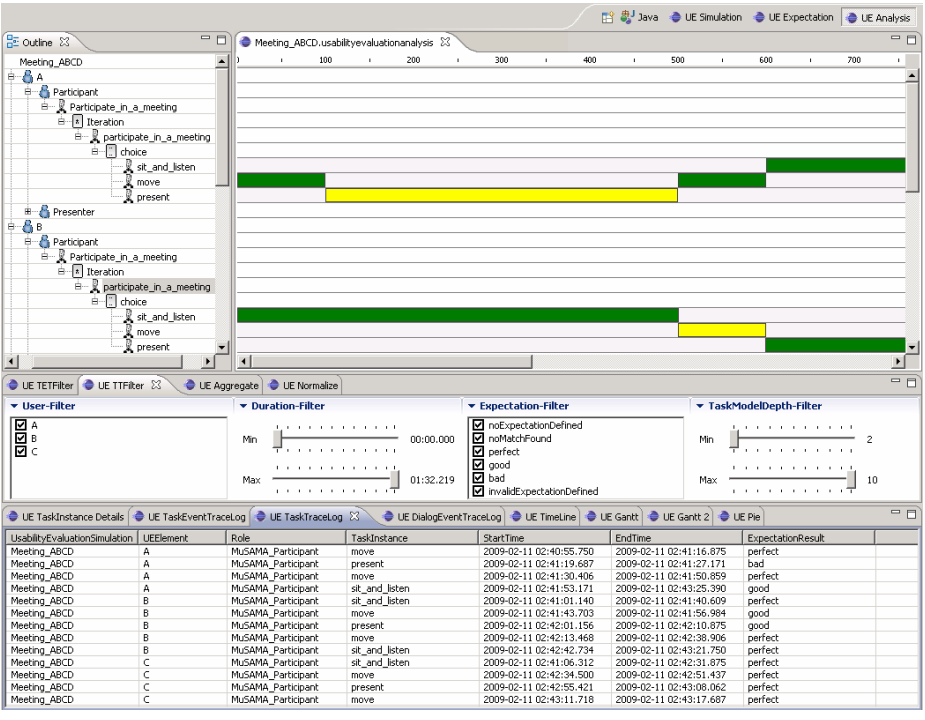

Fig. 4. Analysis

4 Conclusion

In this paper we have enhanced a task modeling notation with a language to express expectations. To ensure a better usability of the specification environment itself we have replaced the first prototypes' textual interface with a GUI to graphically compose expectations in a more convenient way. We have enhanced a task engine to evaluate these expressions during testing. The automatic identification of candidates for usability issues helps to efficiently evaluate more complex systems than supported by existing approaches. While other evaluation approaches only capture and display performed tasks, this paper presented a method to make the implicitly existing expectations explicit and exploit them for usability evaluation.

Future research avenues comprise the evaluation within a field study to discover strength and weaknesses of the approach and incorporate experiences gathered from real world data.

Acknowledgement

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: Jacko, J.A. (ed.) HCI 2007. LNCS, vol. 4550, pp. 1043–1052. Springer, Heidelberg (2007)
2. Hong, J., Landay, J.: WebQuilt: A Framework for Capturing and Visualizing the Web Experience. In: Proc. of the 10th international conference on World Wide Web, Hong Kong, China, pp. 717–724 (2001) ISBN:1-58113-348-0
3. OCL 2.0 specification of the OMG, `http://www.omg.org/docs/formal/06-05-01.pdf`
4. Malý, I., Slavík, P.: Towards Visual Analysis of Usability Test Logs. In: Tamodia 2006, Hasselt, Belgium, pp. 25–32 (2006)
5. Mori, G., Paternò, F., Santoro, C.: CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. IEEE Trans. Softw. Eng. 28(8), 797–813 (2002)
6. 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. 155–169. Springer, Heidelberg (2007)
7. Propp, S., Buchholz, G., Forbrig, P.: Task Model-based Usability Evaluation for Smart Environments. In: Forbrig, P., Paternò, F. (eds.) HCSE/TAMODIA 2008. LNCS, vol. 5247, pp. 29–40. Springer, Heidelberg (2008)
8. Hilbert, D., Redmiles, D.: Extracting Usability Information from User Interface Events. ACM Computing Surveys 32(4), 384–421 (2000)
9. 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
10. Rubin, J.: Handbook of usability testing. In: Hudson, T. (ed.) Wiley technical communication library (1994)
11. Wurdel, M., Propp, S., Forbrig, P.: HCI-Task Models and Smart Environments. In: Proc. of HCIS 2008, Mailand, Italy (2008)
12. Zettlemoyer, L., Amant, R., Dulberg, M.: IBOTS: Agent Control Through the User Interface. In: International Conference on Intelligent User Interfaces (IUI), pp. 31–37 (1999)