Model-Based Specification and Validation of User Interface Requirements

Birgit Bomsdorf[1] and Daniel Sinnig[2]

[1] Department of Applied Computer Science,
Fulda University of Applied Sciences, Germany
[2] Department of Computer Science and Software Engineering,
Concordia University, Montreal, Quebec, Canada
birgit.bomsdorf@hs-fulda.de, d_sinnig@cs.concordia.ca

Abstract. Core functional requirements as captured in use case models are too high-level to be meaningful to user interface developers. In this paper we present how use case models can be systematically refined into detailed user interface requirements specifications, captured as task models. We argue that the transition from functional to UI specific requirements is a semi-formal step which necessitates experience, skills and domain knowledge of the requirements engineer. In order to facilitate the transition we sketch out an integrated development methodology for use case and task models. Since the engineer is also responsible for establishing conformity between use cases and task models we also show, how this validation can be supported by means of the WTM task model simulator.

Keywords: Requirements specification**,** use case model, task model, model simulation.

1 Introduction

A common challenge in Software Engineering (SE) as well as in Human-Computer Interaction (HCI) is the transition from functional requirements to user interface (UI) specific requirements. UI development and the engineering of functional requirements are still often carried out by different teams using different processes and lifecycles [1]. This is likely to result in duplication of effort, inconsistencies, and even contradicting requirements. The apparent gap between software engineering and UI development has been noted by several authors [2, 3, 4] and has been (partially) addressed in our work [5] and the work of others [6, 7, 8, 9].

It has been noted that one possibility to close this gap is to conceptually join use case models and task models in one common development process. The functional requirements of the application are captured by the use case model, which are then stepwise refined into UI-specific requirements captured by task models. A combination of both models has been first investigated by Paternò [10]. In his work, however, the transition from use case to task specifications is performed informally, and task modeling is part of the design process and is not considered at the requirements level. Kujala [9] defines a systematic process for transforming user needs into use case

J.A. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2009, LNCS 5610, pp. 587–596, 2009.
© Springer-Verlag Berlin Heidelberg 2009

specifications, but does not take into account task model specifications. Sinnig et al. [5] have defined a common semantic model for use case and task models, and propose a formal, *but static*, refinement relation between the two artifacts. We firmly believe that the requirements engineer should not be exempted from deciding whether or not a task model faithfully refines the use case it is developed from. On the contrary, finding the answer often depends on domain knowledge and properties specific to a project. Often refinements validation cannot be automated but has to be carried out manually by the requirements engineers themselves. In such a case, simulation and animation have proven to be powerful tools, assisting the requirements engineer in assessing the validity and accuracy of development artifacts [11, 12, 13].

Based on the discussion above, the contributions of this paper are twofold: (1) We propose a systematic and integrated development process according to which UI requirements are derived as a logical progression from a functional requirements specification. (2) We demonstrate how our tool *WTM Simulator* [12] assists the requirements engineer in verifying whether a task model is a valid refinement of a given use case model.

The remainder of this paper is organized as follows: In Section 2, we sketch out, from a generic point of view, the key characteristics of the development process we propose. Section 3 and 4 define use case models and task models as means for capturing functional and UI requirements, respectively. In Section 5 we introduce the WebTaskModel (WTM) approach and present its application to verifying conformity between use case and task models. Finally, in Section 6 we conclude and provide an outlook to future avenues. Related work is discussed throughout the paper.

2 Systematic and Integrated Development Process

The basic idea of our current work on a systematic and integrated development process is depicted in Fig. 1. Use cases are used to capture the bare functional requirements of the system, which are afterwards refined to UI specific requirements by means of a set of task models. Both use cases and task models belong to the family of scenario-based notations, and as such capture sets of usage scenarios of the system. In theory, both notations can be used to describe the same information. In practice and in our approach however, use case models capture requirements at a higher level of abstraction whereas task models are more detailed. Ideally, the functional requirements captured in use cases are independent of a particular user interface [7, 14], whereas the refined requirements captured in the task models take into account the specificities of a particular type of user interface and the characteristics of a detailed user role. For example, if the application supports multiple UIs (e.g., Web UI, GUI, Mobile, etc.) and multiple user types (e.g., novice user and expert user), then the use case model is instantiated to several task models; one for each “type” of user interface and user.

In modern software engineering, the development lifecycle is divided into a series of iterations. Within each iteration, a set of disciplines and associated activities are performed while the resulting artifacts are incrementally refined and perfected. The development of use case and task models is no exception to this rule. On the one hand, ongoing prioritization and filtering activities during the early stages of development will

gradually refine the requirements captured in the use case model. On the other hand, a task model is best developed in a top-down manner, where a coarse grained task model is gradually refined into a more detailed or more restricted task model. In both cases, it is important to ensure that the refining model is a proper refinement of its relative base model (and all its predecessor models). Validation is an important step of a model-based approach so as to avoid ill-defined or miss-behaving models impacting the final design.

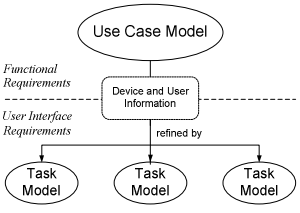


Fig. 1. From Functional requirements to UI Requirements

To illustrate the introduced development process, we use an example that is based on a scenario in which a new web-based Invoice Management System (IMS) is to be developed. It should feature (among others) the following functionalities: "Order Product", "Cancel Order", "View Orders", and "Ship Order". All the functionalities shall be accessible through a *Web UI* and should support two user types: *New Customer* and *Registered Customer*. As a first step, a functional requirements specification in the form of a use case model is developed, which is shown next.

3 Functional Requirements Specification: Use Cases

Use cases were introduced in the early 90s by Jacobson [15]. He defined a use case as a "specific way of using the system by using some part of the functionality." Modern popularizations of use case models are often attributed to Cockburn [14]. Use case modeling is making its way into mainstream practice as a key activity in the software development process (e.g. Rational Unified Process [16]). There is accumulating evidence of significant benefits to customers and developers [17].

A use case model captures the "complete" set of use cases for an application, where each use case specifies possible usage scenarios for a particular functionality offered by the system. Every use case starts with a header section containing various properties (e.g. primary actor, goal, goal level, etc). The core part of a use case is its main success scenario. It indicates the most common way in which the primary actor can reach his/her goal by using the system. A use case is completed by specifying the use case extensions. These extensions define alternative scenarios which may or may not lead to the fulfillment of the use case goal.

An example use case is given in Fig 2. The use case captures the interactions for the "Order Product" functionality of the previously mentioned Invoice Management System (IMS). The main success scenario of the use case describes the situation in which the primary actor directly accomplishes his/her goal of ordering a product. The

extensions specify alternative scenarios which may (3a, 6a) or may not (7a) lead to the abandonment of the use case goal.

In the next section we show how the "Order Product" use case is refined by UI-specific task models.

Use Case: Order Product

Primary Actor: Customer
Goal: Customer places an order for a specific product.
Level: User-goal

Main Success Scenario:

1. Primary actor browses the product inventory and selects a specific product for purchase.
2. Primary actor specifies the desired quantity
3. System validates the availability of the product quantity and displays purchase summary.
4. Primary actor provides/validates payment and shipping information.
5. System prompts primary actor to accept the terms of conditions and to confirm the order.
6. Primary actor accepts and confirms.
7. System has the **payment authorization unit** to carry out payment and finalizes order.
8. System confirms and invoices the order.
9. *Use case ends successfully*

Extension Points:

3a. **The desired product is not available:**
3a1. System notifies primary actor that product in desired quantity is not available.
3a2. *Use case ends unsuccessfully*

6a. **The primary actor cancels the use case:**
6a1. *Use case ends unsuccessfully*

7a.**The payment information is invalid:**
7a1. System notifies customer that payment information provided is invalid.
7a2. *Use case resumes at step 4*

Fig. 2. "Order Product" Use Case

4 Refined UI Requirements Specification: Task Models

Task modeling is by now a well understood technique supporting user-centered UI design. The resulting specification is the primary input to the UI design stage in most HCI development approaches. Since we use task models to refine the raw requirements specification given by use cases, several task specifications may be defined for a single use case, one for each type of user interface and/or user type.

A task model describes how users will be able to achieve their goals by means of the future application. Furthermore it also indicates how the system will support the involved (sub)tasks. Several approaches to defining such models exist (e.g., CTT [13], TaO Spec [18], MAD [19] and VTMB [11]). The WebTaskModel (WTM) used here is a further development of our previous work [11] to account more appropriately for characteristics of interactive web applications. The enhancements, however, are applicable to conventional interactive systems as well. In the following we are not going to point out web-specific details but introduce only those extensions as relevant for this paper. A more comprehensive overview of WTM can be found in [12, 20].

Fig 3. shows a subset of a task model refining the "Order Product" use case described above. The task model was specifically developed for a *Web UI* and the user type *New Customer*. As usual, the task hierarchy shows the decomposition of a task into its subtasks which can be of different task types. In the specification of refined UI

requirements we distinguish between *cooperation tasks* (represented by ⇆) to denote pieces of work that are performed by the user in conjunction with the application, *user tasks* ( ) denoting the user parts of the cooperation performed without system intervention, and *system tasks* ( ) defining pure system parts. *Abstract tasks* ( ), similarly to CTT [13] and MAD [19], are compound tasks whose subtasks belong to different task categories.

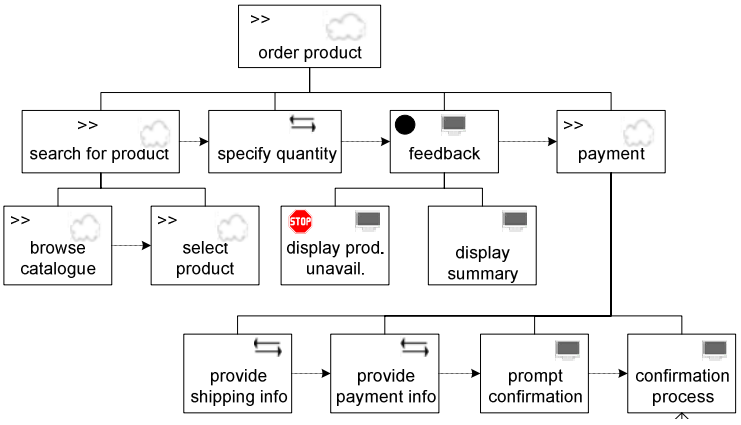


Fig. 3. "Order Product" Task Model for the role New Customer

The order of task execution is given by temporal relations. In the notation used in the figure, temporal relations are denoted by abbreviations: The symbol ● defines a selection of subtasks. >> denotes tasks that are to be performed strictly one after the other in the specified order (visualized by ⋯⋯≫).

The partial task model shown in Fig 3. specifies the task *order product*, which is decomposed into the subtasks *search for product* (according to step S_1 of the use case), *specify quantity* (step S_2) *feedback* (S_3 and S_{3a1}) and *payment* (steps S_4 – S_8). The task *feedback* is decomposed into the subtasks *display summary*, for which we define the precondition `C1:product quantity available`, and *display prod. unavailable*, for which we define the precondition `NOT C1`. Both conditions are derived from the use case extension 3a. Please note that the conditions are not shown in the diagram but were assigned by means of the task property window of the WTM editor (see [20]). The task *display prod. unavailable* is a so-called stop task. It denotes the premature termination of the scenario and is the task model counterpart to use case step S_{3a1}.

In addition to the task model for the role *New Customer*, a task model for a *Registered Customer* is compiled. It differs from the presented task model in terms of how the *payment* task is broken down. Instead of having to provide the shipping and payment information in each case, a registered customer has the option to alter shipping or payment data or to entirely skip the involved subtasks. As seen, different sub-roles lead to slightly different UI requirements. If different UI types were to be supported the use case model would also be refined into device specific task models.

5 Tool Supported Validation

As mentioned above, use case models capture requirements at a higher level of abstraction whereas task models are more detailed taking into account the specificities of a particular type of user interface and characteristics of a detailed user role. The question arises whether or not a task model faithfully refines the use case it is based on. The requirements engineer is not exempted from deciding this question as finding the answer often depends on domain knowledge and project details.

In the following we demonstrate how the tool WTM Simulator [12] can be used to check whether a task model is behaviorally equivalent to a given use case. Firstly, use cases are transformed into a formal (machine readable) presentation based on finite state machines. In the WTM approach, task models are represented by a set of task state machines, which are used within the final application as part of the UI controller [21]. Task state machines are also used to simulate task models within the development steps. In the work reported here a formal correspondence between use case and task models is established to simulate their execution in conjunction. This will be presented by means of a concrete simulation example.

5.1 Mapping Use Cases to UC-FSM

At first use cases are transformed into a finite state machine representation called UC-FSM. A UC-FSM is a labeled, directed, connected graph, where nodes denote states and the edges represent state transitions. In a UC-FSM the execution of a step is denoted by a transition. The transition labels serve as references to the corresponding steps in the original use case description. We believe that UC-FSM capture easily and intuitively the nature of use cases.

As use cases are typically captured in purely narrative form the derivation of the use case graph will be a manual activity. The composition of the use case graph from a given use case depends on the flow constructs, which are implicitly or explicitly entailed in the use case. Examples of such flow constructs are: jumps (e.g. use case resumes at step X), sequencing information (e.g. the numbering of use case steps), or branches to use case extensions. Concrete details on the mapping process as well a slightly more elaborated formal model can be found in [22].

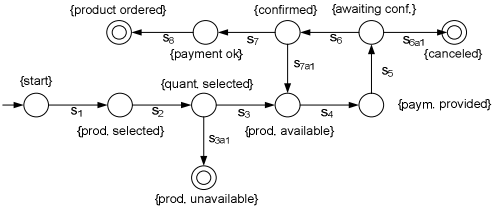


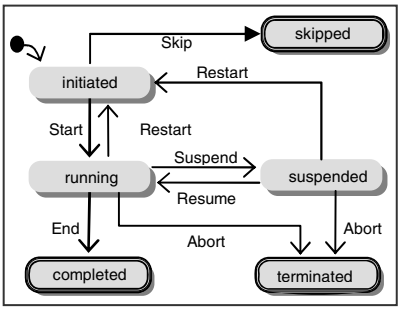
Fig. 4. Use Case FSM for "Order Product" Use Case

Fig 4 depicts the corresponding UC-FSM for the "Order Product" use case. As shown, all the steps of the use case are also present in the UC-FSM. Note that starting from states {quant.selected}, {awaiting confirmation} and {confirmed}, two transitions

are defined, denoting the execution of steps in the main success scenario and alternatively the execution of steps defined in the corresponding extensions.

5.2 Task State Machine and UC-FSM Assignment

In WTM each task formally possesses a state machine describing a generic task life cycle (see Fig 5). For each task the state machine can be extended to specify application specific task behavior. The rules that are used for this purpose are of the form *task.task-state.task-event* → *action*, where *task* denotes the task whose behavior is extended, *task-state* and *task-event* denote the state and corresponding trigger event upon which the *action* is to be performed. In the work presented in this paper, this "extension" technique is used to combine task state machines with the UC-FSM. The objective is to specify dependencies between task executions and use case steps.



State	Meaning
initiated	if all preconditions are fulfilled the task can be started
skipped	the task is omitted
running	denotes the actual performance of the task and of its subtasks, if applicable
completed	marks a successful task execution
suspended	the task is interrupted

Fig. 5. Generic Task State Machine

In order to run a conformance simulation we extend the various task state machines such that they generate the trigger events needed to run the UC-FSM. The specification of the extensions rules depends on which tasks are meant to be a refinement for which use case step. Hereby, due to the before mentioned different levels of abstraction, one use case step is often refined by several tasks.

Table 1 (column 1 and 2) depicts the refinement mapping between use case steps and tasks. Note that *abort order product* is added since S_{3a1} is a stop task. The mappings defined by the row of step S_4 result from the task model differentiation of the role *Customer*. Column 3 of Table 1 depicts the state of the task state machine responsible for sending the corresponding use case event to the UC-FSM. Examples of rules resulting from Table 1 are:

display summary.completed.on_entry → send *S3* to *Use Case product order*
display prod. unavailable.completed.on_entry → send *S3a1* to *Use Case product order*
→ send *abort* to task *order product*

Finally we note that the table is manually created by the requirements engineer. According to our experiences we argue that if the task model was specifically developed based on a given use case specification (as suggested in the paper) the corresponding refinement mappings are clearly defined and hence the conception of the table is a straightforward activity.

Table 1. Refinement Mapping between Use Case Steps and Tasks

Step	Task	Task State
S_1	search for a product	completed
S_2	specify quantity	completed
S_3	display summary	completed
S_{3a1}	display prod. unavailable	completed / *abort* order product
S_4	*New Customer*: provide payment information	completed
	Registered Customer: alter data	completed or skipped
S_5	prompt confirmation	completed
…	…	…

5.3 WTM Simulation Tool and Example

In [12] we presented a tool that supports the developer in validating task, role, task-object models and their behavioral interrelations by means of model simulation. In the tool each task is represented by an icon showing static and dynamic information about the task (such as the task type, temporal relations, and the current state). A context menu attached to each task allows triggering one of the events that are defined by the generic task state machine and are currently valid. The WTM simulator provides the software engineer with different areas implementing several views on the models, e.g., showing the hierarchical task structure, listing all tasks that can be started or ended at a current point in time, respectively, and presenting task objects. Some examples are shown by the screenshots in Fig 6. Here, the object area shows only *USE CASE product order* and its state changes resulting from task execution. Please note that modeling use cases as objects is only a workaround since the use case extensions are not yet implemented in the WTM simulator.

In the upper part of Fig 6 the UC-FSM is in the state *quant.selected*. Since the condition `C1` is fulfilled (see condition area) the task *display summary* can be performed at this point in time. After its completion the UC-FSM state changes to *prod. available* and *provide shipping information* is enabled (indicated by the arrow in Fig 6). The second scenario shows the unsuccessful run in case of `NOT C1` (defined by `C2`): Once the *display*-task is executed *order product* is terminated (thus the startable leaf task area is empty) and the UC-FSM switches to state *prod. unavailable*.

During simulation the requirements engineer can check whether or not each task sequence allowed by the task model is a valid scenario according to the use case specification and vice versa. Furthermore, the simulator allows also one to observe how the steps of a scenario under investigation affect task-objects and domain objects, respectively. As in the case of the *USE CASE* object, the simulator tool represents them in the object area showing their name, classes, and their manipulations in terms of state changes. Similarly, but not depicted in Fig 6, a role area shows all defined roles, allowing the investigation of role changes resulting from task execution as well as disabling and enabling of tasks caused by role changes. For example, the requirements engineer can check the validity of a user registration scenario (by which the role has to change from *New Customer* to *Registered Customer*) and its coactions with the use cases and task models, respectively, defined for each role.

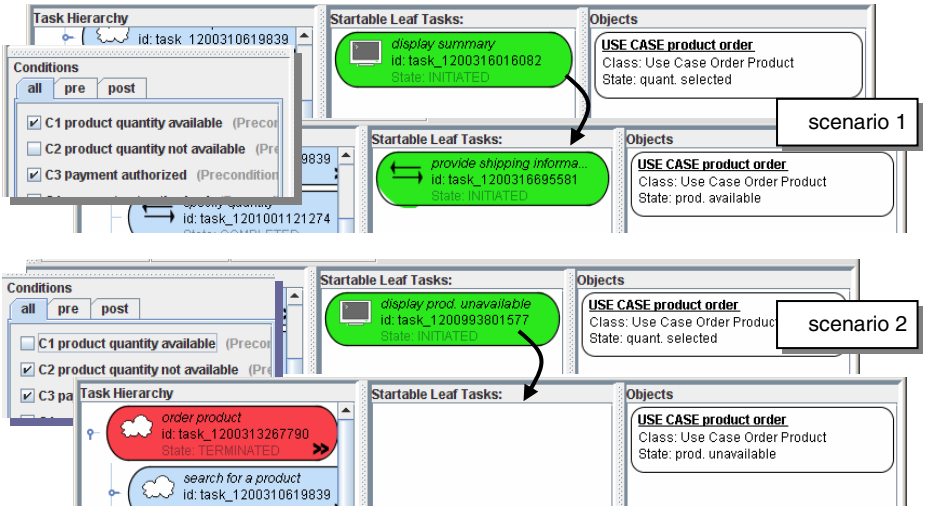


Fig. 6. Simulating Task and Use Case Executions

6 Conclusion

In this paper we presented our current work towards an integrated development methodology for the derivation of UI requirements from high-level functional requirements. The development approach reported here consists of two basic steps. First, a use case model is iteratively created to capture core application requirements. Next, the use case model is successively refined into a set of task models. While use cases capture "raw" functional requirements which are independent of a particular user interface, task models capture refined UI specific requirements which not only take into account the specificities of a particular type of user interface but also the characteristics of a detailed user role. As a result, one use case is typically refined by several task models; one for each UI type or user role. The focus of this paper was on the systematic development of use case and task models. Our approach, however, takes also user roles and involved objects into account - the description of which has been omitted for the sake of conciseness.

The tool WTM Simulator was used to check conformity between a task model and a given use case model. In particular, we demonstrated how use cases can be translated into a state machine representation and formally combined with the task state machine approach of WTM, which in turn is used as input to the simulator. The results of the simulation guide and assist the developer in deciding whether the task model is a valid refinement of the underlying use case.

The research reported in this paper is the first offspring of a larger project, the goal of which is the establishment of a model-driven UI engineering framework, encompassing all phases of the software lifecycle and involved models. Within our next working step we will elaborate the refinement of the functional requirements, e.g., by means of UML activity diagrams. We also aim to further extend the WTM Simulator such that it allows for direct input of structured textual use cases and (semi) automatically generates refinement mappings between use case steps and tasks.

References

1. Kazman, R., Gunaratne, J., Jerome, B.: Why Can't Software Engineers and HCI Practitioners Work Together? In: Proc. of HCI Intern., Crete, Greece, pp. 504–508 (2003)
2. Ferre, X., Juristo, H., Windl, H., Constantine, L.: Usability basics for software developers. IEEE Software 18(1), 22–29 (2001)
3. Kazman, R., Bass, L., John, B.: Bridging the gaps between software engineering and human-computer interaction. In: Workshop at ICSE 2004, Scotland, UK (2004)
4. Sutcliffe, A.: Convergence or Competition between Software Engineering and Human Computer Interaction. In: Seffah, A., Desmarais, M.C., Metzger, M. (eds.) Human-Centered Software Engineering -Integrating Usability in the Software Development Lifecycle, pp. 71–83. Springer, Heidelberg (2005)
5. Sinnig, D., Chalin, P., Khendek, F.: Common Semantics for Use Cases and Task Models. In: Proc. of Integrated Formal Methods, Oxford, England, pp. 579–598 (2007)
6. Clemmensen, T., Norbjerg, J.: Separation in Theory – Coordination in Practice. In: Workshop Bridging the Gap between Software Engineering and HCI, Portland (2003)
7. Constantine, L.L., Lockwood, L.A.D.: Software for Use: A Practical Guide to the Models and Methods of User Centered Design. Addison-Wesley, Reading (1999)
8. Constantine, L., Biddle, R., Noble, J.: Usage-Centered Design and Software Engineering: Models for Integration. In: Workshop Bridging the Gaps Between SE and HCI, Portland (2003)
9. Kujala, S.: Linking User Needs and Use Case-Driven Requirements Engineering. In: Human-Centered Software Engineering-Integrating Usability in the Development Process, pp. 113–125 (2005)
10. Paternó, F.: Towards a UML for interactive systems. In: Nigay, L., Little, M.R. (eds.) EHCI 2001. LNCS, vol. 2254, pp. 7–18. Springer, Heidelberg (2001)
11. Biere, M., Bomsdorf, B., Szwillus, G.: Specification and Simulation of Task Models with VTMB. In: Proc. of Computer-Human Interaction Conference, pp. 1–2 (1999)
12. Bomsdorf, B.: The WebTaskModel Approach to Web Process Modelling. In: Proc. of Task Models and Diagrams for User Interface Design, Toulouse, France, pp. 240–253 (2007)
13. Paternò, F.: Model-Based Design and Evaluation of Interactive Applications. Springer, Heidelberg (2000)
14. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, Boston (2001)
15. Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. ACM Press (Addison-Wesley Pub), New York (1992)
16. Larman, C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd edn. Prentice Hall PTR, Englewood Cliffs (2004)
17. Merrick, P., Barrow, P.: The Rationale for OO Associations in Use Case Modelling. Journal of Object Technology 4(9), 123–142 (2005)
18. Dittmar, A., Forbrig, F., Stoiber, S., Stary, C.: Tool Support for Task Modelling - A Constructive Exploration. In: Proc. of DSV-IS, Hamburg, Germany, pp. 59–76 (2004)
19. Sebillotte, S., Scapin, D.L.: From users' task knowledge to high level interface specification. International Journal of Human-computer Interaction 6, 1–15 (1994)
20. Bomsdorf, B.: Modelling Interactive Web Applications: From Usage Modelling towards Navigation Models. In: Proceedings of 6th International Workshop on Web-Oriented Software Technologies – IWWOST 2007, Como, Italy, pp. 194–208 (2007)
21. Betermieux, S., Bomsdorf, B.: Finalizing dialog models at runtime. In: Baresi, L., Fraternali, P., Houben, G.-J. (eds.) ICWE 2007. LNCS, vol. 4607, pp. 137–151. Springer, Heidelberg (2007)
22. Sinnig, D., Chalin, P., Khendek, F.: LTS Semantics for Use Case Models. In: Proceedings of ACM - SAC 2009, Honolulu, HI (to appear, 2009)