

Chapter 2

MeMo: Usability Workbench

In this chapter, the MeMo workbench for semi-automatic usability testing (Möller et al. 2006) is introduced.¹ The author was a member of the team developing this workbench, and all work presented in this book was largely influenced by the ideas and outcomes of this project.

The MeMo project aimed at the development and implementation of a general user simulation framework, applicable to different kinds and classes of systems, such as SDSs and Graphical User Interfaces (GUIs). A number of requirements for such a framework were set before and during its realization. The workbench should support the simulation of interactions as well as the analysis of the simulation outcome. In addition, it should be easy to use by system designers, as it was realized that many promising approaches to automatic testing are used less than they could just because they require knowledge a designer does not generally have. Thus, the development of MeMo was accompanied by focus groups with designers of spoken dialog systems and web sites.

Inspired by CogTool (John and Salvucci 2005), a tool for assessing simple design drafts with respect to expected task duration, it was quickly realized that including a system model editor in the workbench would provide theoretical and practical advantages. Theoretically, modeling the system provides the user of such a workbench with the possibility to quickly describe a design idea in an illustrative

¹ The description reuses text fragments and figures from (Engelbrecht et al. 2009) and (Engelbrecht et al. 2008a).

way, without using a native programming language.² In addition, the designer can be supported in creating the interaction from the planning stage onwards. As an example, MeMo could prompt the designer to specify a target user group before the system draft can be created, thus forcing adherence to the Usability Engineering Lifecycle (Nielsen 1993). Practically, describing all systems using the same framework allows for a definition of a general, abstract interface to the user model. Like this, the user model is not required to process surface information such as GUI views or natural language prompts.

However, the workbench should extend the capabilities of CogTool in a number of ways, most importantly the integration of suboptimal user behavior due to usability problems. While CogTool aims at estimating the performance (in terms of time) of an expert doing a task without errors, MeMo should

- provide a realistic estimate of the expected performance of one or several target user groups, which may include unnecessary steps,
- allow the detection of problematic system characteristics or dialog steps,
- provide an estimate of the quality of the interaction as it would be rated by the user.

In addition, the system model creation should be simplified, as a plain state machine approach as used in CogTool easily becomes unmanageable given real-life systems.

Simulation of user behavior was chosen as method, as it allows creating a realistic distribution of different user behaviors. As the evaluation should start before test users are confronted with the system for the first time, the simulation should be widely independent of interaction data obtained with the system under test. Instead the user model should be derived from the system description, as well as from general knowledge the designer has about the users and the task. Furthermore, to allow modeling of interaction problems, the user model should produce behavior based on a task description which might differ from the task description of the system, reflecting the notion of a so-called “Mental Model” the user develops when interacting with a system (Norman 1983). Such a task description should comprise what tasks can be done (also called *task model*) as well as how they are performed (also called *interaction model*). Finally, the behavior of the user model should be interpretable to guarantee the control and understanding of its behavior. Only then the designer can derive the steps to be taken for the improvement of the system.

A general user model, automatically generated and applicable to all kinds of systems, is far out of reach. The system designer needs to provide information about the users’ knowledge, their goals, and about their interaction behavior. While intuitively this procedure seems to be logically circular (the designer will always fit the system to the expected needs of the users), Kieras (2003) points out

² In the focus groups, it turned out that most designers use tools which allow quick prototyping, such as PowerPoint, Flash, or UML in case of SDSs.

that a model of the interaction summarizes the design and therefore can be inspected to understand how the design supports the user in performing the task. Furthermore, the complexity of interactive systems nowadays easily exceeds the tracing capabilities of designers. This is especially true for SDSs, where the system has to cope with quasi-random speech recognition or understanding errors. In addition, not all requirements for a design can be fulfilled in all cases, and the designer needs to carefully choose which guidelines to prioritize. In such a scenario, sample dialogs generated by a user simulator can provide valuable hints for design decisions (see e.g. López-Cózar et al. 2003). At a later stage, after first user tests have been carried out, simulation can be useful to re-use corpora for fine-tuning of the system. Still, simulation will not replace user tests completely, but serve as a tool to get the most out of the information available at each design stage.

A further advantage of capturing the designer's knowledge in a model is that the knowledge is then formalized and can thus be shared between different designers. Thus, a requirement for the user simulation was to be as general as possible, and to allow re using knowledge gained from earlier designs. A solution to this was found in allowing the designer to specify rules describing the behavior of users given certain characteristics of the interface. In addition, knowledge about user characteristics, such as age or technical affinity, can be stored as a *user group* (see below), which later can be used for simulations. In the next chapter, it will be shown how rules can be defined for different user groups and system characteristics by looking at an example application. Before this, the basic concepts of MeMo are introduced.

2.1 Model Creation in the MeMo Workbench

2.1.1 System Model

For the system we distinguish a task model and an interaction model. The task model describes tasks which can be conducted with the system by specifying conditions which have to be met for the interaction to be considered successful. These conditions are formulated by specifying the target assignments for variables which are defined for the project, and which can be modified during the simulation (called *information* in MeMo). For example, the “slots” of a dialog system could be modeled as *informations*. Like for many SDSs, *informations* have the form of attribute-value-pairs (AVPs). They can be transferred from the user to the system (that is, constraints are specified) or from the system to the user (i.e., the user acquires knowledge necessary for the solution of the task). We assume this to be a common principle in task-oriented human machine interaction. Therefore, this conception is applied to GUIs and SDSs alike.

The system interaction model describes how the system behaves in the interaction with the user. For this purpose, the system is described as a state machine, in

which the transitions between states can be conditional on the user actions in the states, or on other, freely defined parameters. Furthermore, a transition can trigger consequences affecting the behavior of the system and the user (e.g. a transition could imply that certain constraints are automatically set and thus do not have to be queried by the system).

The states of the model can be composed of voice dialogs or of graphical views, which provide interaction possibilities to the user. In case of SDSs, the possible interactions are fields (or “slots”) which can be filled in the respective system state. For GUIs, interaction elements such as links or buttons are specified. Dialogs and views can also be attributed with features of the prompts (e.g. information conveyed in the prompts, dialog act, prompt length) or of the display (e.g. information conveyed by labels of interaction elements or other text on the screen, sizes and positions of interaction elements).

These features determine the conditions under which the user model chooses its path through the system interaction model. The designer can choose freely which aspects of the system are described with features. These can vary depending on the aspects the simulation is focusing on. However, some important features (e.g. the degree of openness of system questions) are queried by the workbench during model building or are automatically acquired (e.g. prompt length).

Figure 2.1 shows a screenshot of the Dialog Designer. From the left panel, a view can be selected. The top-right panel shows the currently selected view. In case of a GUI, a sketch or a screenshot would be presented here, on which interaction elements can be marked and displayed. On the bottom panel, features and settings for the view can be defined.

The system model editor is shown in Fig. 2.2. In the top mid panel, the views of the currently selected state are displayed. Note that a state can contain several views (e.g. a voice interface and a GUI). Views can be selected from the left panel. On the right hand side, all states are listed. To connect the current state to a target state via an interface element, a line can be drawn from that interface element to the state. The transition would then appear in the bottom mid panel. Clicking a transition opens the transition editor, in which conditions and consequences can be specified.

2.1.2 User Model

Like the system model, also the user model distinguishes between a task model and an interaction model. A *user task model* corresponds to one of the system tasks, but contains the user’s task knowledge, a start state, and goal conditions. The users’ task knowledge would typically be the constraints to be communicated to the system, specified as *informations* (i.e. AVPs). The start state specifies which state is presented to the user at the beginning of the interaction. The goal conditions are equivalent to those in the system task, but it would be possible to model different goal conditions for the user, e.g. based solely on *informations* visible to

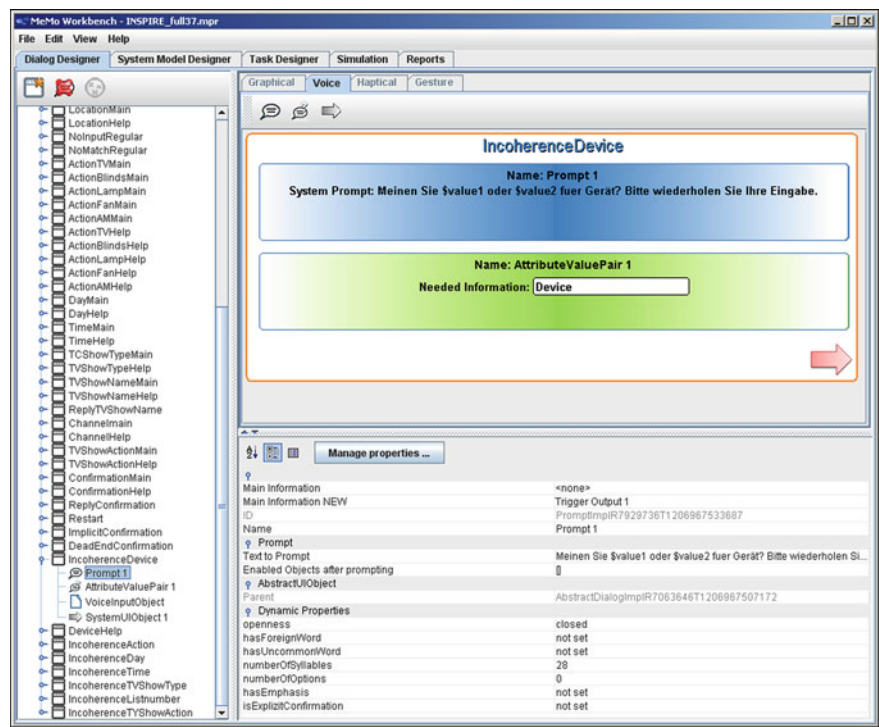


Fig. 2.1 Dialog designer of the MeMo workbench. The *top-right panel* shows the currently selected view. In case of a GUI, a sketch or a screenshot would be presented here, on which interaction elements can be marked and displayed. On the *bottom panel*, attributes and settings for the view can be defined

the user. By formulating the goal in the form of conditions, we allow the user model to ignore errors with negligible consequences for the user. E.g. if a user wants to switch on two out of three lamps with a smart home system, she might regard the interaction successful if all three lamps are switched on. Figure 2.3 shows how MeMo supports the creation of the task models for system and user. At first, conditions determining task success as concerned by the system are specified. Then, the user task editor can be opened, showing the conditions assumed by the user model to be preconditions to task success. In addition, a start state can be selected, and the user model is equipped with task knowledge, i.e. a set of constraints leading to the task goal.

The *user interaction model* is responsible for simulating interactions with the system. At each state, the user model searches for interaction possibilities which allow conveying constraints from the task knowledge. In an SDS model, an interaction possibility is equivalent to a slot which can be filled at that state. The search for possible interactions is enabled by those annotated features in the system model which concern the semantics of the dialogs or views. Thus, the

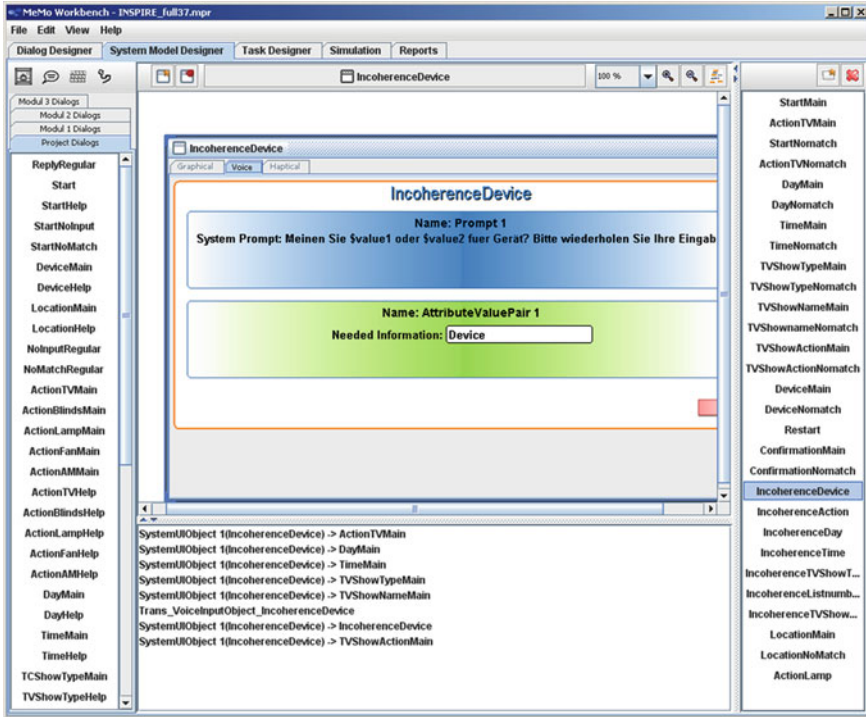


Fig. 2.2 System model editor, including a list of views (left), a list of states (right), views in the currently selected state and transitions leading out of this state

respective features of the system model (e.g. button labels) must match the *information* names. If no matching *information* can be found in the current state, it is assumed that a user would look for interaction possibilities which are semantically related to the task constraints. For example, if the task is to look for a restaurant serving pizza and the system offers to select Italian or French food, the user would be more likely to choose Italian, as pizza is a meal from the Italian cuisine. To model this, a *keyword engine* can be used to calculate the semantic similarities between the labels of interface elements and the task constraints. The required information about semantic relations between terms can be extracted from web dictionaries, as described in detail in (Steinökel et al. 2011).

Next, probabilities are calculated for each interaction possibility, under consideration of the features of the state and its interaction elements. Initially, the selected transition is associated with the highest probability. However, the probabilities of all possible interactions are then varied by rules which take as conditions specific sets of system model features and their values, as well as user characteristics. By this, deviations from the direct path to the goal can be simulated.

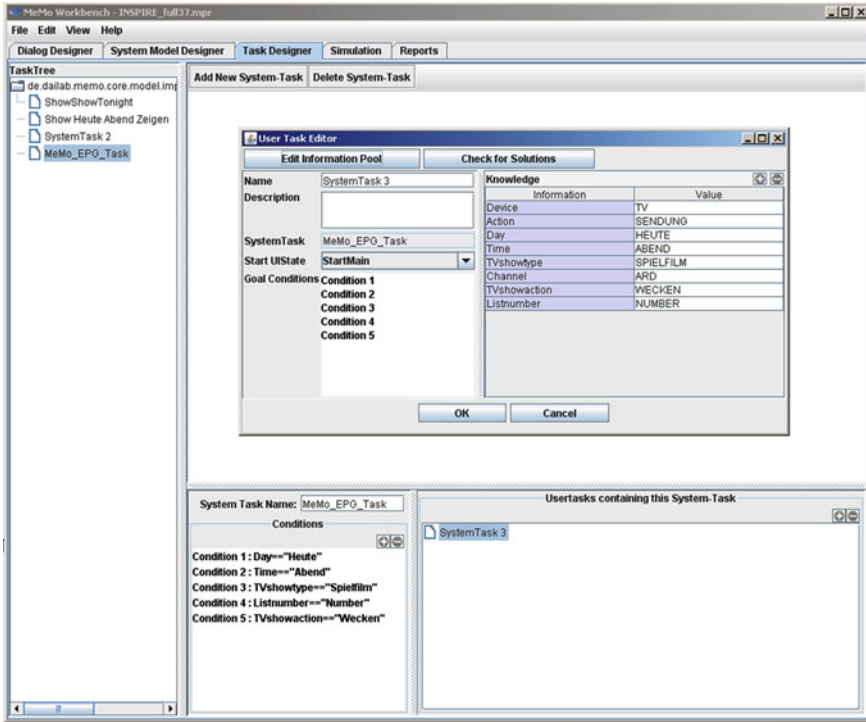


Fig. 2.3 The two windows of the task editor, on top the user task editor, which allows selecting goal conditions and defining task knowledge. The *bottom window* shows a list of tasks in the *left panel*, system task goal conditions on the *bottom left*, and a list of user tasks. The *top window* was opened by clicking on a user task

At each interaction step, the rule engine evaluates the features of the current system state and the user characteristics, selects the rules which apply, and calculates the resulting probabilities. Changes in probabilities are specified only approximately, as rules should be as general as possible. A rule can increase or decrease the probability for a specific interaction by an amount specified abstractly using symbols as placeholders. Concrete values for each symbol are specified in a configuration file. This way, by changing the configuration file, the effect of all rules can be adjusted to reflect user behavior as closely as possible.

While general rules applicable to all interfaces or to all interfaces of one class may be possible, a designer can also custom-tailor rules for the current simulation in order to modify the behavior of the modeled users. This way, simulations can be run under specified assumptions, such as expected user errors (cf. e.g. López-Cózar et al. 2009). The content of the rules can be derived from expert knowledge, observations in experiments, or usability heuristics. All rules include a description which helps the designer to understand why rules have been triggered in the simulation.

2.1.3 User Group Editor

Simulations can be run for different groups of users, as it was previously done by Eckert et al. (1997). These authors define four very distinct user groups with extreme behavior patterns, each group reflecting a single psychological trait (e.g. “patient” users hang up the phone after 99 turns). Janarthanam and Lemon (2008) simulate the behavior of users groups with different domain knowledge in a troubleshooting system. The groups are basically distinguished by asking for extended help messages more or less often.

The user group editor of MeMo tries to go beyond these approaches by allowing the definition of more detailed user groups. Each user group is described by a set of characteristics which are considered to be relevant for user interaction behavior. The user characteristics considered in MeMo have been derived from previous work on the classification of users (Hermann et al. 2007). The cited paper aims at a user classification scheme which is based on interaction behavior, thus allowing recruitment of a comprehensive set of users for usability tests. To do this, the most differentiating characteristics of users were collected at first. Examples are affinity to technology, anxiety, problem solving strategy or domain expertise. In the user group editor, also the user’s age and deficits like hearing impairment can be specified.

A user group is defined by assigning a value to each characteristic given in a GUI editor (Fig. 2.4). All characteristics can have a neutral value (or range), which means that this characteristic is not considered in the calculation of the probabilities for the user actions. Like this, the knowledge about users can be formalized in a user group, while not all characteristics have to be known to the person defining the user group.

To generate differentiated user behavior, in addition to the user groups rules need to be defined which describe the relation between user characteristics and user behavior. Note that rules are not specified for the groups, but for the characteristics and their values. E.g. a rule might state that users with high anxiety more often fail to utter their request in time. Such relations are much easier to estimate than the behavior of a complex user with many characteristics.

To instantiate a user for an iteration of the simulation, it is sampled from the user group by assigning corresponding values for each characteristic. If a range of values is specified for a group, a concrete value out of that range is selected randomly.

2.1.4 Speech Understanding Error Simulation

A further module integrated into the MeMo workbench simulates speech recognition and understanding errors to be encountered in real-life systems. As a naming convention, the term “speech understanding” (SU) will be used in this book for the combination of speech recognition and language understanding. A number of

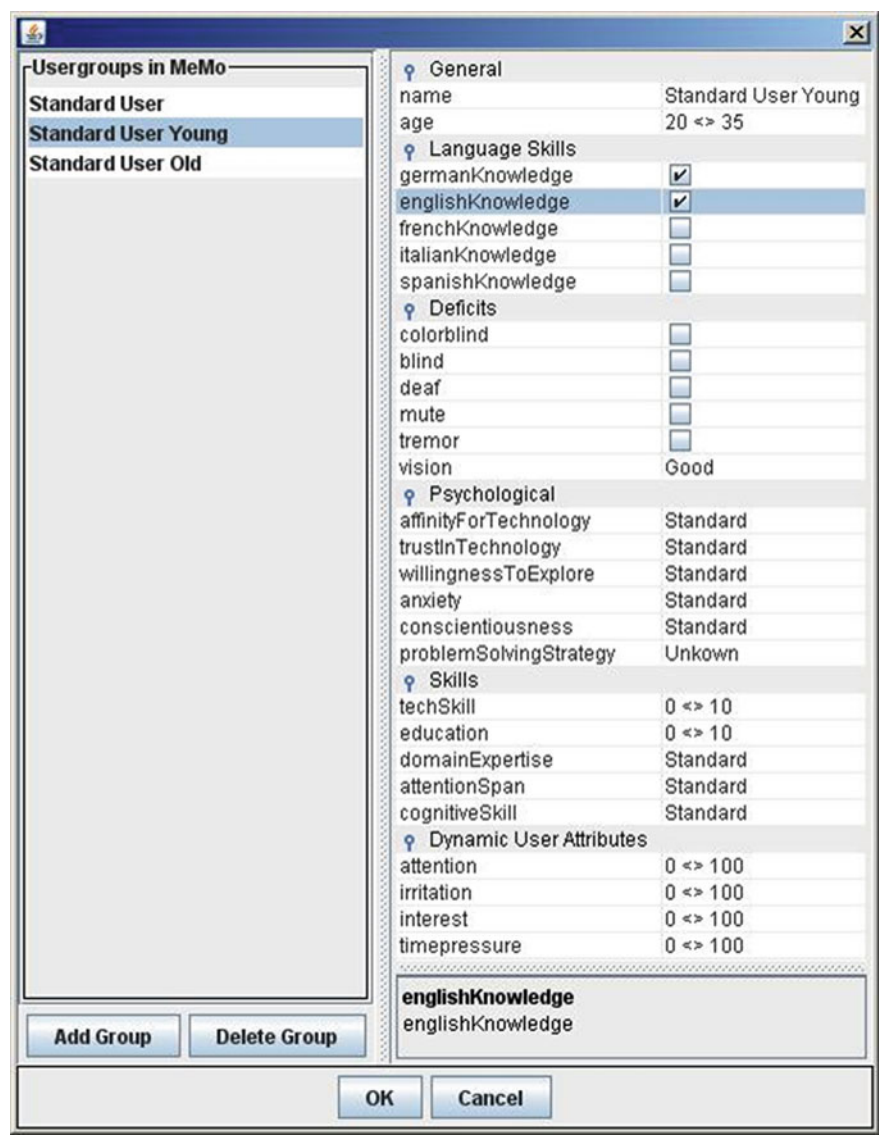


Fig. 2.4 User group editor of the MeMo workbench

approaches to model SU errors have been proposed in the past. To cite just a few, a simple model by Pietquin and Renals (2002) estimate the error probabilities from the type of recognition task, while a more complex approach might take into account acoustic confusability of potential user utterances, as in Schatzmann et al. (2007c). We chose a simple model, in which deletions, substitutions and insertions

of semantic concepts are generated at random to an amount specified in a configuration file. The amount of such errors may also depend on system and user characteristics, such as prompt openness or domain expertise, and can therefore be adapted according to the current situational context and user type during the interaction. In early development phases, before the ASR and language understanding units are fully developed, such an approach can model the knowledge which the developer has about the system and its potential users reasonably well. For later phases, a more complex error model could be integrated into the simulation process.

2.2 Reporting

2.2.1 *Formative Usability Report*

The designer gains access to detailed data from the simulation by an interactive graph, which displays each state the simulated user passed through, and each chosen interaction (Fig. 2.5). Furthermore, the graph highlights problematic states, e.g. states which deviate from the shortest goal-driven path are marked orange, while optimum states are marked green. By selecting a transition its probability, triggered rules and an estimated time prediction for this interaction are made available to the designer. Especially the triggered rules and their description in addition to the cause of the triggering event, e.g. a button is in the wrong place, hint towards steps to take to improve the system's usability.

In addition to the inspection panel, the workbench allows exporting the interaction data as Excel sheets, so they can be inspected manually.

2.2.2 *Summative Report*

The designer might be interested in the overall results of the simulation, especially if several system versions or user groups are compared. These are output in a PDF document including a description of the simulation settings and figures showing the results. The figures are laid out to compare results for different user groups. Moreover, an Excel sheet can be generated containing the dialog-wise measures.

In addition to performance measures regarding effectiveness and efficiency, the workbench foresees the prediction of a user satisfaction score from the log files. Unfortunately, so far there exists no general model for the estimation of perceived quality from interaction data. Thus, at the moment this function is not included in the workbench. However, for experiments with the workbench, a prediction algorithm can be trained on experimental data similar to those simulated. Like in Walker et al. (1997), Linear Regression can be used for such models. More

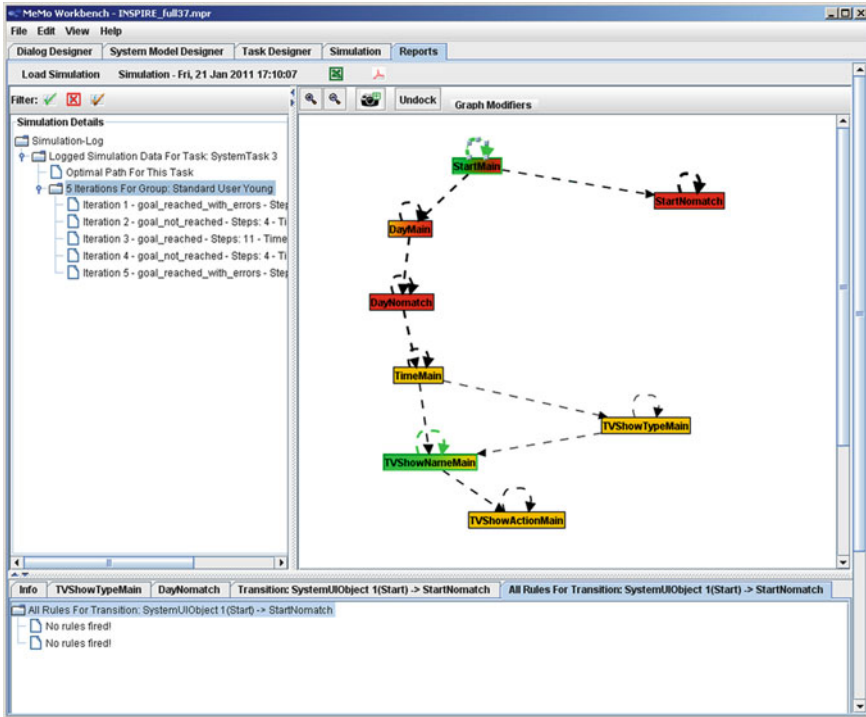


Fig. 2.5 Formative usability report, including a panel to select interactions (*left*), a panel showing a state graph representation of the interaction (*right*), and a panel where additional information about the interaction can be displayed (*bottom*)

sophisticated models are discussed later in this book. As a deliverable of the MeMo project, it could be demonstrated that such a model is feasible also for web sites (Engelbrecht et al. 2008b).

Although such models usually show low accuracy in terms of R^2 , especially for the prediction of independent data, previous work showed that the prediction of mean judgments for system configurations, aimed at here, can be quite accurate (Engelbrecht and Möller 2007).

2.3 Chapter Summary

In this chapter, we introduced the MeMo workbench for semi-automatic usability evaluations. The workbench can be controlled through a GUI and supports all steps from model creation to analysis of the simulation results. The development drew on research in human–computer interaction and spoken dialog systems and it was

grounded on feedback from its potential users. In the next chapter, we examine the validity of the user simulation integrated in the workbench. In particular, we show how an experiment can be conducted with MeMo, and discuss how the results can be evaluated with respect to their validity.



<http://www.springer.com/978-3-642-31590-9>

Estimating Spoken Dialog System Quality with User
Models

Engelbrecht, K.-P.

2013, XIV, 130 p., Hardcover

ISBN: 978-3-642-31590-9