# Looking for Unexpected Consequences of Interface Design Decisions: The MeMo Workbench

Anthony Jameson[1], Angela Mahr[1], Michael Kruppa[1],
Andreas Rieger[2], and Robert Schleicher[3] ⋆

[1] German Research Center for Artificial Intelligence
[2] Technische Universität Berlin — DAI-Labor
[3] Deutsche Telekom AG Laboratories

**Abstract.** This paper discusses and illustrates work in progress on the MEMO workbench for early model-based usability evaluation of interface designs. Characteristic features of the workbench include (a) the prediction of errors via rules that refer to user attributes; and (b) the automatic generation of methods for performing specific tasks and for recovering from errors.

## 1 Introduction and Example

This short paper discusses work in progress in the project MEMO, which is an effort at T-Labs, a research division of Deutsche Telekom, to introduce task modeling into the design and development process for interactive systems. The MEMO workbench ([1]) is intended to enable the evaluation of the usability of new designs in an early phase of the design process.

Before discussing the salient features of the MEMO workbench on a general level, we will introduce a simple example that will make the subsequent discussion easier to follow. Our minimal example system is inspired by a part of a web site that allows a user to determine, for a given telephone rate package, how much it costs to make a phone call to a particular foreign country. The example system as modeled in MEMO offers information only on the 5 countries shown in the top screen shot of Figure 1. Together with the screens labeled "2" and "3", this screen illustrates the correct method for finding the rate for a call to the United States of America: The user clicks on the radio button for that country and then clicks on the "Next" button, which takes her to a screen showing the desired rate. The two lower right-hand screens ("4" and "5") show the analogous sequence when the country clicked on is the United Arab Emirates.

One topic of interest for MEMO is the prediction of possible errors and their consequences. Even with the simple first task, the user could commit a *description error* ([2]), clicking on the first country in the list whose name starts with "United". This error takes her to State 4, which is off the correct path for her task. At this point she might notice that the wrong country has been selected, in which case she can easily get back onto the correct path by clicking on "United States of America". But if she instead proceeds to click on "Next", she will end up looking at an incorrect rate (State 5).

⋆ The research described in this paper is being conducted in the context of the project MEMO, which is funded by Deutsche Telekom AG Laboratories.

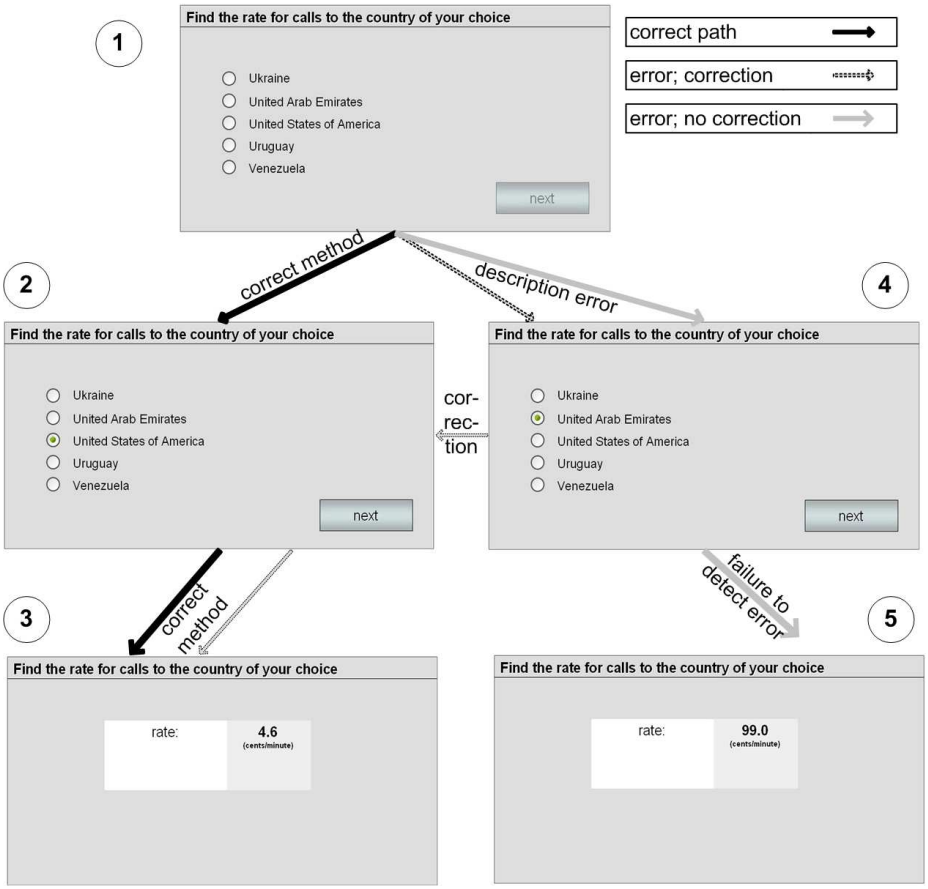


**Fig. 1.** Screen shots illustrating the possible system states that can be reached in the simple example used in this paper as well as the possible paths through these states during the performance of the task of finding the rate for phone calls to the United States of America.

## 2 Goals of MeMo and Relationship to Previous Work

In this simple example, the overall purpose of the MEMO workbench is to allow the interface designer to predict how often each of the paths shown in Figure 1 will be taken by users who are performing the task of finding the rate for the United States—different predictions being made for each combination of user attributes such as visual acuity and the amount of attention devoted to the task.

More generally, MEMO is intended to allow the interface designer to compare a number of alternative designs for a given interface in terms of the likely behavior of users on a specified set of tasks given different combinations of attributes.

We can locate MEMO in the space of existing approaches to model-based evaluation by mentioning some important sources of inspiration. A number of aspects of the MEMO workbench were inspired by COGTOOL (see, e.g., [3]). Both COGTOOL and MEMO enable an interface designer to (a) construct a medium-fidelity prototype of each of several variants of a to-be-designed system; (b) specify how users are likely to interact with each variant of the system while performing specified tasks; and (c) run simulations to predict certain aspects of the users' performance on these tasks (e.g., execution time).

Instead of aiming to match COGTOOL's sophisticated prediction of execution times, MEMO aims at a more explicit and automated prediction of error-related behavior: Errors are generated during simulations by rules that aim to capture known types of error. In this way, MEMO builds both on well-known taxonomies and analyses of human error (e.g., [2]; [4]) and on recent efforts to use these concepts to predict errors in the context of model-based evaluation (see, e.g., [5]; [6]; [7]; [8]). Relative to most such approaches, MEMO tries to automate the prediction of errors to a greater extent, as opposed to relying on human judgment for the specification of likely errors. It is clear that there are limits to such automation, but it seems worthwhile to explore these limits.

In a similar vein, another salient feature of MEMO is the use of automatically computed methods for performing particular tasks—and recovering from errors—as an alternative (or complement) to methods that are explicitly specified by an analyst.

For the realization of the vision sketched so far, a number of questions need to be dealt with. In the following sections, we discuss the approach to each question that is being taken with MEMO.

## 3 What User Attributes Should Be Distinguished?

Even if an interface design is generally successful, it may be problematic for users who have particular (combinations of) attributes (e.g., low visual acuity combined with a limited knowledge of English). Especially problems that are likely to arise only given a combination of two or more attributes may be hard to discover without a systematic, exhaustive search through the space of combinations. In MEMO workbench, various aspects of the simulation of users can be made to depend on such attributes, which can include: perceptual and motor capabilities (e.g., visual acuity); relevant prior knowledge and experience (e.g., amount of experience with systems like the one under consideration); and temporally variable factors (e.g., the amount of attention that the user is devoting to the performance of the task).

## 4 How Is the System Design to Be Specified?

For reasons that will become clear below, in MEMO each system is modeled with a state diagram, as in COGTOOL ([3]). Figure 2 illustrates how a designer can model a system in terms of (a) drawings of individual screens, each of which contains one or more widgets; and (b) transitions between screens that are made when certain actions are performed with the widgets. As is well known, this type of modeling works much better for some types of system than for others.

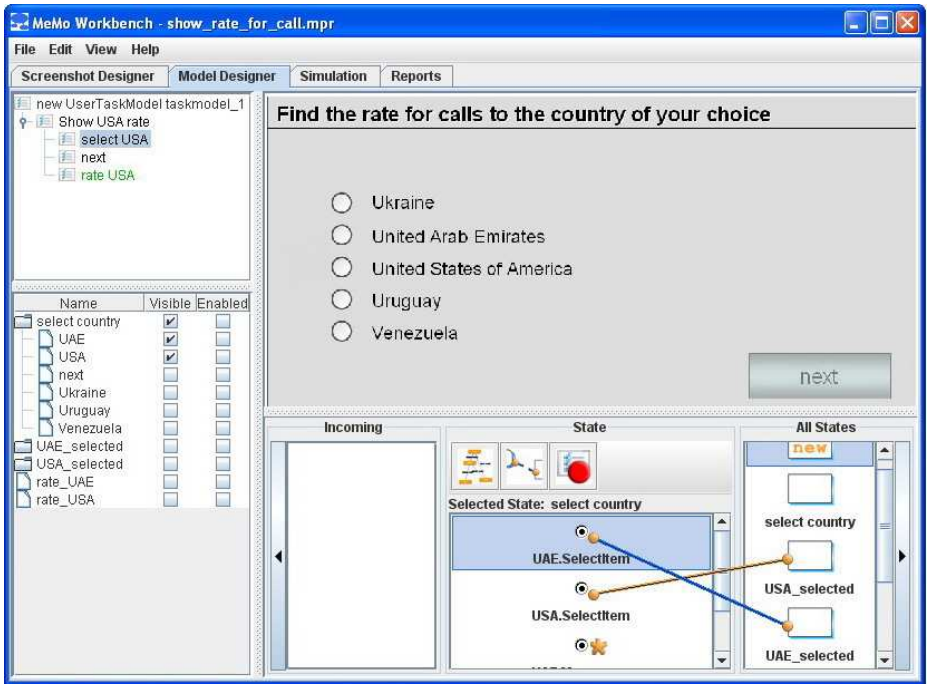


**Fig. 2.** Screen shot of the MEMO workbench's interface for specifying a system variant along with ideal methods for the performance of tasks with that system variant.

## 5 How Are the Tasks and Ideal Methods to Be Specified?

It is assumed that the designer wants to simulate users' performance on a number of tasks for a number of system variants. A typical approach in model-based evaluation is to specify somehow a correct or "ideal" method of performing each task for each variant and then perhaps to characterize various deviations from this method (see, e.g., [8]).

One way of specifying an ideal method for a task (realized, for example, in COGTOOL) is to demonstrate a relevant sequence of steps by operating the relevant widgets in the system model; and the simple method that can be seen in the upper left-hand window of Figure 2 was in fact defined in this way. This approach can become tedious or even impractical, however, when a large number of tasks and (similar) system variants are considered.

An alternative approach that we are currently pursuing is available if each task can be specified in terms of an initial state and a goal state (e.g., States 1 and 3 in Figure 1). In this case, the problem of finding an ideal method for the task can be seen as the problem of finding a good route from the initial state to the goal state within the state diagram, where the goodness of a route depends on properties such as the number of steps or the predicted total execution time. To be sure, a method automatically derived in

this way may not be realistic for all users. For example, the quickest way to accomplish a particular task in a complex commercial website may be to go to the site map and click on one of the hundreds of links found there—a method unlikely to be applied by most users. We therefore expect that the automatic generation of methods will have to be subjected to some constraints, both general ones and constraints for users with particular attributes (e.g., the constraint that keyboard shortcuts are not employed by users who lack previous familiarity with the system in question).

In this way, the generation of an appropriate method for a given task is analogous to the problem of finding a route with a navigation system from a starting point to a destination; and the imposition of constraints on the nature of the methods is analogous to the use of constraints such as "no highways".

## 6 How Should the Basic Simulation Process Work?

As was mentioned above, the basic goal of MEMO is to predict what will happen when a user with certain attributes performs a certain task with a particular variant of the system. For the moment, we assume for the sake of exposition that the simulated user always performs the task according to the ideal method that has been derived for users with the attributes in question; the simulation of errors will be discussed below.

For a given system variant, the generation of simulation runs proceeds as follows:

1. The designer specifies a set of *user groups* for which the simulation is to be carried out, each user group being defined in terms of a combination of values for particular attributes.

2. The designer lists the tasks for which the simulation is to be carried out.

3. The designer specifies a certain number of simulation runs for each user group and each task.

4. In each simulation run, the system generates a trace by assuming that the simulated user applies the ideal method for the user group in question.

5. Once all of the simulation runs have been completed, the system generates a report on the results for each user group and task. In the case considered so far, where no errors are simulated, this report reflects aspects of performance such as the time required by each user group to perform the task and the frequencies with which particular types of operation (e.g., clicking on icons) are performed.

In the case of our simple example, the error-free simulations simply reflect the fact that, for all user groups, each of the possible tasks is performed straightforwardly with two mouse clicks.With realistically complex systems and tasks, however, this simulation approach can yield some interesting results. For example, it may turn out that the ideal method for a particular task involves an unacceptably large number of operations (of a certain type) for at least one user group (e.g., a group that is assumed always to use menus rather than keyboard shortcuts).

## 7 How Should the Workbench Predict Errors?

One way of modeling behavior that involves errors (used, e.g., in COGTOOL) is to treat a method that contains an error simply as one possible method for performing the task.

The remarks made above about the limitations of the manual specification of correct methods apply to an even greater extent here: Once errors are considered, the number of possible methods for performing a task becomes very large, especially since errors can occur in combination.

The approach currently being explored in MEMO is to use a set of general error generation rules to produce incorrect behavior at various points during a simulation: The general procedure for simulating the performance of a given task is to assume that the user will perform the correct next step unless an error generation rule applies to the situation, in which case an error is generated with a probability specified by the rule. In our introductory example, the following rule will generate a description error in some of the simulation runs:

- If the correct action is to select the item $I$ with the label $L$,
- and there is another item $I'$ whose label begins with the same word as $L$,
- then the user will select $I'$ with a probability of $p_1$ if the user's attention to the task is low and $p_2$ if it is high.

Even this highly simplified rule captures the important fact that this error can occur and that it is more likely under certain conditions than under others. The introduction of error generation rules affects the generation of simulation runs as follows:

Whenever the simulated user enters a given state, the workbench checks whether there is an error generation rule that applies in that state (taking into account the next action specified by the ideal method currently being applied by the simulated user). If so, with a probability specified by that rule, the incorrect action prescribed by the rule is simulated, and the system enters a state that is not on the ideal path for the task in question.

We still need to deal with the question of the extent to which errors are detected and recovered from and the consequences that they have.

## 8 How Should the Workbench Predict Error Recovery?

For the sake of exposition, we assume for the moment that the user will do the right thing as soon as an error has occurred: recognize the error and recover from it in the most straightforward possible way.

When an error step is predicted during a simulation run, the MEMO workbench in effect views the user as being confronted with a new task which in general overlaps partly with the original task: The user's task is now to recover from the error and then proceed towards the original goal state. More concretely, the workbench computes on the fly an appropriate method for getting to the goal state starting from the state that resulted from the error; in doing so, the workbench uses the same algorithm that is used for generating ideal methods in the first place.

In our simple example, the workbench's reporting would reveal that, in the simulation runs that contained a description error, the user would quickly recover simply by clicking on the correct country.

If the workbench operated in exactly this way, it would of course yield overly optimistic predictions, assuming optimal error recovery behavior in all cases. Still, the

reporting would contain some useful information. For example, a comparison between the simulation runs that contained errors and those that did not might reveal that all of the predictable errors can be straightforwardly recovered from as long as they are detected immediately—or at the other extreme, it might reveal cases in which no recovery at all was possible. Still, it should also be possible to simulate cases in which the user does not detect an error.

## 9 How Should the Workbench Simulate Failed Error Detection?

On the whole, the question of when users will recognize that they have made an error is a complex one (see, e.g., [5]). MEMO's current approach to error detection is applicable in cases where detection of an error by the user is in principle so straightforward that failure to detect the error can be viewed as an error in itself.

As an illustration, consider our simple example: Once a user has mistakenly clicked on "United Arab Emirates", the screen shows a filled radio button next to the unintended country; so if the user quickly checks the result of her action before clicking on "Next", she will see the need to do exactly what the MEMO workbench predicts according to the principles described in the previous section: Click on the radio button next to "United States of America" and then proceed.

The user can fail to notice her error if she doesn't bother to check but just proceeds to click on "Next". This pattern of omitting a verification step can be modeled roughly with a rule such as the following:

- If on the current screen item $I'$ is marked has having been selected
- and the item that really ought to be selected is some other item $I$
- and there is a button $B$ that the user can click on to proceed to the next screen
- then the user will (incorrectly) click on $B$ with probability $p_1$ if the user's attention to the task is high and $p_2$ if it is low.

Like the first error-generation rule introduced above, this one is hard to formulate in such a way that (a) it applies with some generality and (b) the probabilities $p_1$ and $p_2$ are empirically reasonably accurate. Still, even a rough formulation can lead in our example to the useful prediction that some users—especially those with low attention to the task—will end up in an incorrect final state (i.e., looking at an incorrect rate)—provided that they clicked on the wrong country in the first place. Given that the first error was likewise more probable given low attention to the task, the MEMO workbench will predict a nonnegligible frequency of ending up in the wrong state only for users who show low attention to the task.

Note that, in a different but analogous setting, the first error might be likely given user attribute $A$ (e.g., poor knowledge of English) while the second one was associated with some completely different attribute (e.g., poor visual acuity). In this case, the workbench would predict a nonnegligible frequency of ending up in the wrong state only for users who have *both* of the problematic attributes—thereby uncovering an undesirable outcome that would be hard to detect without systematic search through a large number of attribute combinations and simulation runs.

In sum, this approach to the modeling of (the lack of) error detection is applicable only in cases where errors are basically easy to detect. But it does help call attention to the subset of these cases in which an error is committed and not detected, so that the implications of these cases can be contemplated by the designer.

## 10 Conclusions and Current Work

Some of the characteristic features of the MEMO approach appear to work quite naturally for some types of system, task, and error and less well for others: the representation of a system with a state diagram; the automatic derivation of ideal methods for performing tasks; the rule-based prediction of errors and error detection; and the dependence of predicted behavior on user attributes. We have argued that, where applicable, these features of MEMO make possible some useful types of simulation and analysis that go beyond what is possible with user testing, inspection-based evaluation, and other types of model-based evaluation. The special promise of these features lies in the ability of the MEMO workbench to search systematically through a large space of possibilities that is defined by different system variants, different tasks, different user attributes, and the nondeterministic occurrence of errors. The simulations generated in this way can hardly be as accurate as those yielded by more focused, hand-crafted simulation models, but they may have a greater ability to uncover potential problems that arise only in certain specific situations.

## References


1. Möller, S., Englert, R., Engelbrecht, K., Hafner, V., Jameson, A., Oulasvirta, A., Raake, A., Reithinger, N.: MeMo: Towards automatic usability evaluation of spoken dialogue services by user error simulations. In: Proceedings of INTERSPEECH 2006, the Ninth International Conference on Spoken Language Processing, Pittsburgh, PA (2006)
2. Norman, D.A.: Design rules based on analyses of human error. Communications of the ACM **26** (1983) 254–258
3. John, B.E., Salvucci, D.: Multipurpose prototypes for assessing user interfaces in pervasive computing systems. Pervasive Computing **4**(4) (2005) 27–34
4. Reason, J.: Human Error. Cambridge University Press, Cambridge, New York (1990)
5. Wood, S.D., Kieras, D.E.: Modeling human error for experimentation, training, and error-tolerant design. In: Proceedings of the Interservice/Industry Training, Simulation and Education Conference, Orlando, FL (2002)
6. Paternò, F., Santoro, C.: Preventing user errors by systematic analysis of deviations from the system task model. International Journal of Human-Computer Studies **56**(2) (2002) 225–245
7. Baber, C., Stanton, N.A.: Task analysis for error identification. In Diaper, D., Stanton, N., eds.: The Handbook of Task Analysis for Human-Computer Interaction. Erlbaum, Mahwah, NJ (2004) 367–379
8. Bastide, R., Basnyat, S.: Error patterns: Systematic investigation of deviations in task models. In Coninx, K., Luyten, K., Schneider, K.A., eds.: Task Models and Diagrams for User Interface Design. Springer, Berlin (2006) 109–121