Automated Usability Evaluation during Model-Based Interactive System Development

Sebastian Feuerstack[1], Marco Blumendorf[1], Maximilian Kern[1], Michael Kruppa[2], Michael Quade[1], Mathias Runge[1], and Sahin Albayrak[1]

[1] DAI-Labor
Technische Universität Berlin
Ernst-Reuter-Platz 7,10587 Berlin
Firstname.Lastname@DAI-Labor.de
[2] Deutsches Forschungszentrum für Kuenstliche Intelligenz GmbH
Trippstadter Str.122, 67663 Kaiserslautern
Firstname.Lastname@dfki.de

Abstract. In this paper we describe an approach to efficiently evaluate the usability of an interactive application that has been realized to support various platforms and modalities. Therefore we combine our Multi-Access Service Platform (MASP), a model-based runtime environment to offer multimodal user interfaces with the MeMo workbench which is a tool supporting an automated usability analysis. Instead of deriving a system model by reverse-engineering or annotating screenshots for the automated usability analysis, we use the semantics of the runtime models of the MASP. This allows us to reduce the evaluation effort by automating parts of the testing process for various combinations of platforms and user groups that should be addressed by the application. Furthermore, by testing the application at runtime, the usability evaluation can also consider system dynamics and information that are unavailable at design time.

Keywords: model-based user interface development, automated usability evaluation.

1 Introduction

User interfaces (UI) are more and more required to support several contexts-of-use. They need to be able to be run on several platforms, consider different types of users and adapt to various usage situations. This poses new challenges when it comes to the development of interactive applications as well as their evaluation. In this work we present our approach combining a model-based runtime system with an Automated Usability Evaluation (AUE) tool to provide the ability to evaluate UIs that adapt at runtime. In order to attend to these issues we combined two approaches: The Mental Models (MeMo) workbench, a workbench for AUE and the Multi-Access Service Platform (MASP), a model-based framework for UI generation. Model-based UI development approaches [1, 2, 3, 4] already support the generation of multi-platform user-interfaces as well as context-of-use adaptation. They contain semantics stored in

P. Forbrig and F. Paternò (Eds.): HCSE/TAMODIA 2008, LNCS 5247, pp. 134–141, 2008.

a well-structured form of declarative design models. This allows tools to assist developers at design-time by detecting questionable features and by offering the help of automated advisors. However, most of these approaches do not consider ad-hoc adaptation to the context-of-use which can only be calculated at runtime. The MASP allows the derivation of a UI from a set of executable models [5], defining the user interface state. Besides having the possibility to describe adaptive UIs, the models and the state information can also be utilized to support the AUE of the UI. By working with abstract UI models, which ideally contain all required concepts, the UIs could be generated for any platform and thus usability evaluation could be done by considering any platform without being forced to redefine the concepts of the evaluation target.

After we give an overview of related work in the research field of automated usability evaluation in the next section, we elaborate on the combination of the two approaches. We illustrate the results of the evaluation using the interactive Cooking Assistant we developed based on the MASP (the Cooking Assistant has also been deployed as a demo application in our Ambient Assisted Living Testbed [6]) and we conclude with a summary and outlook in section 5.

2 Related Work

AUE methods can support the evaluation process, whereas they differ significantly in their degree of automation and the effort for evaluators [7]. The majority of AUE methods is usually applied on already existing systems or prototypes and therefore requires re-constructing a system interaction model by reverse-engineering or manually annotating the semantics of already existing applications [8, 9, 10]. The CogTool [11] is a tool to predict execution times for certain tasks. The max model [12] is considering cognitive aspects via user simulation for measuring accessibility of information within web sites and the PROSKIN project [13] is tracking user data in order to aggregate it to higher-level profiles to gain personalized UI designs. The AIDE [14] tool focuses on organizing the controls of an interface by incorporating five metrics (efficiency, alignment, horizontal balance, vertical balance and constraints) into the design process, while initial automated assistance has been proposed by USAGE [15]. Furthermore, tests have been developed for certain aspects of completeness, consistency and command-reach ability [16]. Model-based interface development environments, such as TADEUS's [17], support simulation and model-checking by translating the dialog model into a Petri net.

In contrary to these approaches we are moving the AUE from design-time to runtime in order to enable the evaluation of ad-hoc context-of-use adaptations as well as considering system dynamics that are unknown at design-time, such as data queries. We use the MeMo workbench [18] to simulate different user profiles that perform certain tasks and benefit in the way that usability issues are uncovered for a wide range of possible users. Further on, we can simulate users performing more errors as usual to diagnose the system's behavior which is difficult to predict by real persons within complex systems. The evaluation process of the MeMo workbench is based on a cognitive walkthrough (CWT) carried out by a usability expert and includes a rule engine which contains a set of modifier rules extracted from the CWT methodology.

Compared to the related work, our approach offers the following benefits:

- No need for a manual re-creation of the specification of an application which is fragile for introducing misunderstandings and incompleteness.
- All combinations of platforms and users that are addressed by the interactive application can be efficiently targeted to an automated user interface evaluation.
- By utilizing the model-based run-time system, the evaluation can consider system dynamics and parameters, and context-of-use variations.

3 Model-Based Automated Usability Evaluation

In order to automate the usability evaluation for several context-of-use scenarios, each specifying a combination of a specific platform, a certain group of users and conditions of the environment, we replaced the system interaction model (SIM) of the MeMo workbench with the runtime models of the MASP. The simulation starts with the MASP generating the initial representation of the user interface for a certain context-of-use. This representation is sent to the MeMo workbench by delivering a system interaction state (SI state) which consists of the current enabled set of interaction input and output tasks. Based on this information and the current user profile, the MeMo workbench chooses an interaction which is related to an input task of the MASP. The correlated user action will be performed and a new SI state is generated until the user's goal has been accomplished.

3.1 System Interaction State Generation

In the MASP we are interpreting task trees that define the temporal relations as the basic interaction flow for the interactive system. A domain model completes the task model by providing content for the tasks. The model defines the data structures and holds instances of these structures which are objects that become accessible at runtime. The life-time of these objects is determined by the task model, which also references the objects in the designated tasks as we described in detail in [19]. Modifying objects of the domain model happens either through the service model (1), connecting backend services to application tasks (2) or by user interaction (3) while entering and changing information. User interaction is mediated by the interaction model, detailing the interaction tasks (4). Here we distinguish input interaction tasks (IIT) and output interaction tasks (OIT), which identify the interaction on the highest level of abstraction. While OITs require no human intervention but present information to the user until they become disabled by another task, IITs require human intervention such as data input. The tasks are also annotated with the objects that are read, modified, created or declared and refer to the related classes of the domain model. A reification of the interaction in terms of details is provided by the interaction model. It encloses an abstract interaction description that is modality independent and a concrete interaction description that adds the modality dependent information. Additionally, by mappings between the interaction and the layouting model (5) presented in [20], absolute positions and element sizes of the concrete interaction objects are calculated based on the context model (6) and filled with information delivered by various sensors at runtime. Thus, each SI state will be composed of encapsulated sets of enabled tasks. Each

atomic task is linked to the concrete domain object class and the relevant class attributes which are presented in the interface together with the expected user operation (read, create or modify) (3). Further, the relevant interaction model elements and the calculated layouting information are added for each enabled task of the SI state (5). Finally a cascading style sheet defining the style for the graphic elements of the concrete model is attached to the SI state.

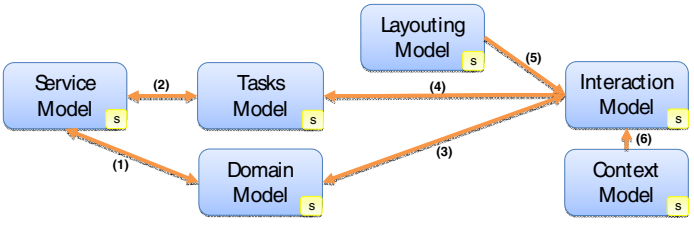

Fig. 1. Overview of the involved runtime MASP models, separating multiple levels of abstraction. Each model comprises (part of) the runtime state(s).

The relation between the models is formalized by mappings which support the exchange of information between the models and keep the models synchronized. Figure 1 shows the different models and how they are related. At runtime each model contains additional state information, allowing the derivation of the final UI according to the state of the application. Through the utilization of mappings between the models, we support an information flow at runtime, which allows identifying active elements of the models based on the enabled task set. The derived SI state allows the creation of a final UI which can be evaluated by an AUE tool as described in the next section.

From the AUE perspective, the runtime interpretation of the UI models creates a SIM containing SI states, which describe the interface and interaction capabilities of the evaluation target. The UI elements contained in a SI state are communication elements with different interactions, e.g. a button can be used with left click or right click as input interaction, whereas the information of the caption is carried through an output interaction from the system to the user. Every input interaction and the successive SI state, which is enabled after the interaction, are encapsulated in a transition to the subsequent SI state. In the simulation phase the SIM is mapped to a dynamic SIM representation which is handed over to the simulator for interaction with the user interaction model (UIM). The simulator continuously presents SI states to the UIM and maps the input interactions, chosen by the UIM depending on the current user task, to the corresponding transitions of the SIM.

3.2 User Action Generation

After the MASP presents the SI state to MeMo, the information processing unit (IPU) of the UIM evaluates all input and output interactions and manipulates the data storage which is used by the planning unit for the final interaction selection process. Both modules are influenced by two types of user attributes (UA). Static UA express the characteristics of a user group and cannot be influenced by the simulation process, whereas dynamic UA (DUA) are flexible properties of a user.

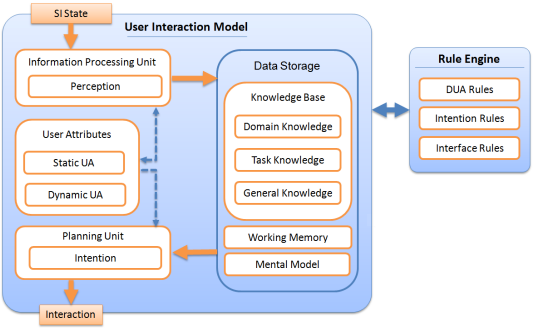

Fig. 2. The components of the user interaction model and the interaction selection process

As shown in figure 2 the knowledge of the UIM is divided into three categories. While the task knowledge describes the knowledge which has to be exchanged to successfully accomplish a task, general knowledge describes knowledge all user groups have in common and domain knowledge depends on the system to be evaluated, e.g. experts know where to look for required information in the system.

During the analysis process, the UIM has to handle four different situations: receiving new information (1), already known information (2), known information with differing values (3), or a request for information (4). Situations 1 and 2 are easy to handle. Either the UIM already knows what the system is presenting or the UIM stores the presented information in its data storage. Situation 3 is a mismatch between the knowledge of the UIM and the presented information by the system. The UIM can react in different ways. If the mismatch refers to the task knowledge, the UIM stores the wish for correction in its working memory, otherwise its domain or general knowledge will be corrected. If the requested information in situation 4 is contained in its knowledge, the UIM is able to respond to the request.

After analyzing the output interactions, the rule engine is called to modify the dynamic user attributes according to DUA rules. The DUA influence the intention of the UIM, e.g. high frustration or irritation level increase the probability of a task abortion, time pressure reduces the probability for requesting help. Four intentions are implemented so far: *forward*, *cancel*, *help* and *abort*. If the UIM has the intention *forward*, it tries to transfer its task knowledge to the system and therefore prefers interactions which support the transfer. In case that no interaction is preferred the UIM tries to navigate in order to analyze further states. If the UIM is navigating and believes further navigation is not possible or reasonable the intention turns to *cancel* and the UIM tries to navigate back. In case it could find neither interactions for knowledge transfer nor meaningful interactions for navigation, the intention becomes *help*. The intention *abort* leads to an abort of the current task.

After the intention alteration the UIM starts to evaluate the available input interactions. Assuming the intention is *forward,* the UIM tries to transfer its task knowledge to the system and therefore prefers interactions which support the transfer. These interactions are set up with a higher probability of selection. In case that no interaction was preferred by the IPU, the planning unit systematically evaluates the UI

objects for navigational functionalities (navigation objects) in correlation with its knowledge and increases their interaction selection probability. If the intention is not *forward*, the UIM tries to find navigation objects related to its intention, e.g. the intention is *help* it prefers buttons labelled with "?", "i" or "Help". In case that no interaction related to *help* or *cancel* can be identified, the intention will be set to *abort* and the UIM gives up. Finally each interaction is set up with a probability of selection, whereas the preferred interactions have higher probabilities than alternative interactions. In order to modify these probabilities, the rule engine analyzes the complete SI state and generates facts for numerous attributes of the UI objects. A dice throw selects the interaction according to the probabilities. The selected interaction is given back to the MASP which generates a new SI state. The interaction selection process will be started again until the task is finished or the UIM aborts the process.

4 Evaluation Process and Results

All data collected during the simulation is captured with the help of a logging module and stored in log files. With the help of an internal view the designer gains access to simulation details for each task, user group and iteration of the simulation. This way, the designer can retrace every interaction the UIM has chosen. This information consists of the interaction element, the list of triggered rules, the probability distribution of the available interactions and further statistics, e.g. execution time. A portion of these data is visualized within an interactive graph (compare figure 3). Each node of this graph represents a SI state which has been passed by the UIM and each transition represents the chosen interaction. Deviations from the shortest goal driven path are highlighted in different colors This helps the designer to easily uncover problematic SI states and to find reasons for the deviations, because in the current state of implementation the workbench is not offering this level of critique by itself. As described in section 1 the MeMo workbench evaluated the interactive Cooking Assistant (CA) which is presented to the workbench by the MASP. In the following, evaluation results of several simulation runs are described.

With the help of the rule engine, several limitations of the CA regarding its usability were exposed. Figure 3 illustrates a problematic iteration in which the UIM accidently chose a different meal from a list of presented meals and did not discover this fact immediately in the state *RecipeDetails*. In fact the probability of this interaction is low, but the consequences might cause costs for the user, e.g. buying ingredients for a different recipe as intended. In the subsequent state a dialog in which the number of persons to whom the meal should be prepared for was presented to the UIM. In this state the name of the meal is not displayed and therefore the UIM did not discover its wrong meal choice. The same problem occurs in the state *ShoppingList*, where the UIM can decide whether to continue with the CA or prepare a shopping list with the necessary ingredients. Finally the UIM is able to discover its wrong meal choice in the last state of figure 3.

In this dialog the information of the chosen meal is displayed and returned to the user as output interaction. The only input interaction to undo the selection is via a link

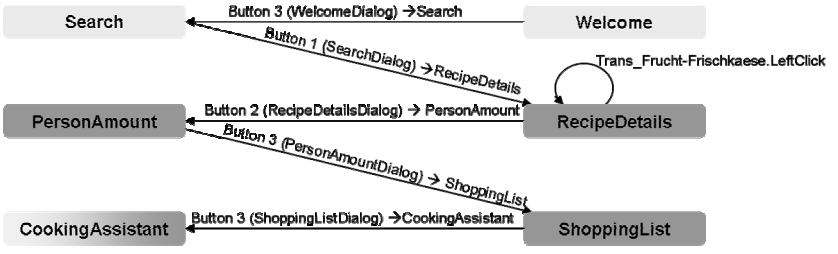

Fig. 3. An interactive graph which displays user interactions and presented SI states

to the start page in the upper left corner of the screen. As a result of its non-conform coding as a link it might not be recognized as an interaction element and the rule engine therefore reduces its interaction probability. Within a couple of iterations the UIM did not choose this interaction which has the consequence that the user finds no proper interaction to correct the meal selection and therefore aborts the task.

Another result of the evaluation was the low contrast of the font compared to the underlying interaction elements and furthermore the low contrast of some buttons compared to the background, e.g. a small white font was used on light blue buttons. The rule engine discovered this fact and reduced the interaction probabilities for user groups representing elderly people with poor vision. These user groups had a higher probability of not finding the appropriate interaction element and therefore choosing an alternative interaction that deviated from the shortest goal driven path.

Finally, using standard labels for some interaction elements (e.g. “Next”) to achieve higher coherences could improve the usability as well, which was confirmed by further simulations done with the help of the workbench.

5 Conclusion and Outlook

In this paper we presented our approach to utilize user interface design and runtime models to support an AUE of the modeled system. Utilizing the models allows skipping the usually required manual annotation of the final UI with the underlying design concepts. The approach also makes the evaluation of multi-platform and context adaptive user interfaces much more straightforward because the automated system is able to make assumptions about context parameters or used platforms. Thus it can evaluate multiple variants of the UI much easier. We have also shown the evaluation of one of our applications which revealed several usability problems.

For the near future we plan to extend the approach to consider more details of the UI descriptions as well as to consider multimodal aspects of the UI and extended context information (e.g. voice-based feedback could help noticing the selection of a wrong recipe earlier). We would like to support the evaluation of different context situations, e.g. simulating migrated or distributed UIs. Along with the challenge, we would also like to enrich the perception of the UIM by concerning further mental aspects. Finally our goal is to automatically make constructive suggestions (beyond critique) for improving the usability.

References

1. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interacting with Computers 15(3) (2003)
2. Coninx, K., Luyten, K., Vandervelpen, C., Van den Bergh, J., Creemers, B.: Dygimes: Dynamically generating interfaces for mobile computing devices and embedded systems. In: Mobile HCI (2003)
3. Mori, G., Paternò, F., Santoro, C.: Design and development of multidevice user interfaces through multiple logical descriptions. IEEE Trans. Softw. Eng. 30(8) (2004)
4. Reichard, D., Forbrig, P., Dittmar, A.: Task models as basis for requirements engineering and software execution. In: Proceedings of TAMODIA (2004)
5. Blumendorf, M., Lehmann, G., Feuerstack, S., Albayrak, S.: Executable Models for Human-Computer Interaction. In: Proc. of DSV-IS (2008)
6. Blumendorf, M., Feuerstack, S., Albayrak, S.: Multimodal smart home user interfaces. In: Proc. of IUI4AAL Workshop on IUI 2008 (2008)
7. Ivory, M.Y., Hearst, M.A.: The state of the art in automating usability evaluation of user interfaces. ACM Comput. Surv. 33(4) (2001)
8. Tarta, A., Moldovan, G.: Automatic Usability Evaluation Using AOP. In: IEEE International Conference on Automation, Quality and Testing, Robotics, vol. 2, pp. 84–89. IEEE Computer Society, Los Alamitos (2006)
9. Ritter, F.E., et al.: High-level behavior representation languages revisited. In: ICCM (2006)
10. Paternò, F., Piruzza, A., Santoro, C.: Remote web usability evaluation exploiting multimodal information on user behaviour, pp. 287–298. Springer, Heidelberg (2006)
11. Teo, L., John, B.E.: Comparisons of keystroke-level model predictions to observed data CHI 2006: CHI 2006 extended abstracts on Human factors in computing systems, pp. 1421–1426. ACM Press, New York (2006)
12. Lynch, G., Plamiter, S., Tilt, C.: The max model: A standard web site user model. In: 5th Conference of Human factory & the Web (1999)
13. Fine, N., Brinkman, W.: EUSAI 2004, pp. 15–18. ACM, New York (2004)
14. Sears, A.: Aide: a step towards metric-based interface development tools. In: UIST (1995)
15. Byrne, M.D., Wood, D., Sukaviriya, P.N., Foley, J.D., Kieras, D.E.: Automating interface evaluation. In: CHI Conference Companion (1994)
16. Braudes, R.E., Sibert, J.L.: Conmod: a system for conceptual consistency verification and communication. SIGCHI Bull. 23(1) (1991)
17. Elwert, T., Schlungbaum, E.: Modelling and generation of graphical user interfaces in the tadeus approach. In: DSV-IS (1995)
18. Jameson, A., Mahr, A., Kruppa, M., Rieger, A., Schleicher, R.: Looking for unexpected consequences of interface design decisions: The memo workbench. In: Winckler, M., Johnson, H., Palanque, P. (eds.) TAMODIA 2007. LNCS, vol. 4849, Springer, Heidelberg (2007)
19. Feuerstack, S., Blumendorf, M., Albayrak, S.: Prototyping of multimodal interactions for smart environments based on task models. In: European Conference on Ambient Intelligence: Workshop on Model Driven Software Engineering for Ambient Intelligence Applications (2007)
20. Feuerstack, S., Blumendorf, M., Schwartze, V., Albayrak, S.: Model-based layout generation. In: Proc. of Advanced Visual Interfaces (2008)