Automatic Adaptation of User Workflows within Model-Based User Interface Generation during Runtime on the Example of the SmartMote

Kai Breiner[1], Kai Bizik[1], Thilo Rauch[1], Marc Seissler[2], Gerrit Meixner[3], and Philipp Diebold[1]

[1] Software Engineering Research Group, University of Kaiserslautern, 67663 Kaiserslautern, Germany
[2] Center for Human-Machine-Interaction, University of Kaiserslautern, 67663 Kaiserslautern, Germany
[3] German Research Center for Artificial Intelligence (DFKI), 67663 Kaiserslautern, Germany
{Breiner,K_Bizik,T_Rauch,p_diebol}@cs.uni-kl.de, Gerrit.Meixner@dfki.de, Marc.Seissler@mv.uni-kl.de

Abstract. Model-based universal interaction devices are already capable to react on contextual changes by automatically adapting the user interface, but without considering the usefulness of the resulting user interface. Often tasks cannot be executed any more or execution orders will result in dead locks caused by unavailable functionality. We present our approach of investigating this property of adapted models based on the example of the SmartMote in our living lab the *SmartFactory*$^{\mathrm{KL}}$. Given the task description of the user interaction we determine a dialog model in terms of a state machine – which is necessary in our process of user interface generation – to determine possible execution orders leading to the accept state of this state machine. Using these execution orders the initial task model can be adapted, all misleading tasks can be removed and the resulting user interface will only offer valid user interactions.

Keywords: Adaptive User Interfaces, Usage Context, Task Fulfillment, SmartFactory, SmartMote, Model-based User Interface Development.

1 Introduction

Modern production environments comprise numerous devices demanding for direct human interaction – e.g. to execute, monitor, or maintain the production process. While each of this devices confront the user with different user interfaces – providing different user experience – future facilities will be equipped with universal control devices such as the SmartMote (see Figure 1, right) that are capable of providing control over intra-device functionality in a homogeneous manner, enabling for a homogeneous user interaction concept. This can be ensured by the separation of the content (i.e. task/workflow model) and its representation (i.e. presentation model). Due to the flexible nature of future ambient production environments, production lines can be (automatically) rearranged e.g. in order to compensate for device errors.

Thus, devices and their services will not be available anymore which needs to be reflected by the user interface in order to prevent usage errors. Therefore, the underlying model will be tailored to the new configuration. This paper discusses one approach based on the adapted model to determine if the user is still able to achieve her goal.

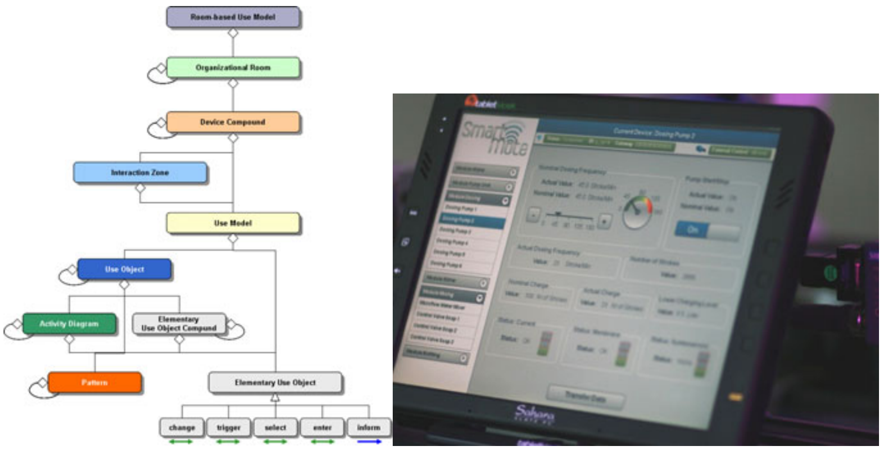

Fig. 1. Left: the room-based use model and its components. Right: the SmartMote running on a TabletPC used to access devices of the *SmartFactory*$^{\mathbf{KL}}$.

2 SmartFactoryKL and SmartMote

Located in Kaiserslautern, the *SmartFactory*$^{\mathbf{KL}}$ serves as a living lab for future ambient production environments [15]. It is characterized by a vast amount of field devices from various vendors. Basically, these devices build up a production process with the aim to produce colored soap filled up in dispenser bottles. According to the vendor, physical constraints, and device type these field devices offer diverse ways of human interaction – ranging from simple digit numbers to complex industrial workstations. Many of these devices possess their own user interfaces which – according to the vendor or physical device restrictions – provide different interaction concepts and can result in a bad user experience. The user has to adapt herself to different concepts, cooperate designs or modalities. Being confronted by this diversity during the execution of workflows this will result in operating errors [3].

Future production environments will consist of flexible entities which are replaceable and reconfigurable, but also aware of the entire process. In case of a device error during the production these lines will be able to reconfigure themselves in order to still be able to complete the production process. According to the current configuration the user has to adapt her mental model of the entire production flow in order to be able to locate the interface she has to access. For maintenance personnel this can be a challenging task.

Furthermore, the *SmartFactory*$^{\mathbf{KL}}$ implements the paradigm of new ambient-intelligent production environments all of the field devices are interconnected and can provide their information at any time, which is a necessary requirement to develop universal interaction concepts. Given the combinatorial explosion of possibilities, traditional software development approaches are insufficient to develop such flexible user interfaces. Therefore, our universal interaction concept – the SmartMote – is based on various formal descriptions that can be altered and interpreted during runtime to produce a user interface giving access to the desired functionality of the field devices.

Core model of our approach is the room-based use model (RUM) which is built on the Useware Markup Language (useML) [5]. Figure 1 (left) depicts the RUM as it is used in our approach. It consists of a topological model of the entire environments comprising physical/organizational spaces (rooms) that can include devices. These devices are equipped with interaction zones describing the concrete restrictions on human-machine-interaction with respect to the current usage situation. Use model objects (UO) describe the tasks and workflows that can be executed by the user by a hierarchy down to elementary use objects (eUO) that stand for the individual user interaction such as change, trigger, select, enter and inform. Because the RUM only provides information about the content to be displayed there are many other aspects it does not cover. To interface to concrete application services the functional model is used [4]. We made use of the concept of usability patterns [2] to assign the elements to be displayed to concrete user interface elements in a presentation model.

In the following we will focus on the RUM – especially the contained task model – to show how the current usage situation has an influence on the content to be displayed and the effect on a user of the SmartMote.

3 Context in Ambient Intelligent Production Environments

Knowledge about the current usage situation can be used to improve the usability and perceived experience in various ways. During the interaction with the system the intention of the user can be predicted and the system can prepare itself to support future task executions. If the user will be within the operating range (e.g. during maintenance) of a robot arm while activating it this will lead to a hazardous situation. In case the system is able to detect the users' current position all functionality leading to this problem can be deactivated in advance and feedback can be provided to inform the user.

Context is a very broad term of which many definitions exist [12]. In general it summarizes all surrounding circumstances of a certain user interaction or activity. It contains every piece of information that can be used to characterize the environment in which the interaction takes place, attributes of the user as well of the devices the user is interacting with which additionally is relevant to this particular interaction [14]. Collecting this data in a formal model will create a knowledge base that can be used in order to derive facts about the current usage situation based on which the user interface can be adapted (e.g. deactivation of hazardous tasks). Thus, the information which is being provided to the user can be adapted to the current usage situation and the complexity of functionality offered to the user can be reduced to the functionality that is relevant in this situation and therefore usage errors can be prevented.

By the application of a set of rules on the contextual information will result in the generation of transformations as well as filters that will be applied to the RUM – certain tasks will not be available any more to the user. Therefore, the system needs to perform task reachability analysis in order to determine if the user can still reach her goal: completing the workflow. This aspect will be covered in section 5.

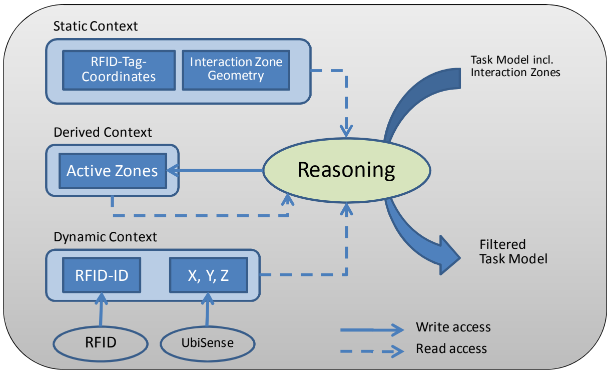

Fig. 2. Processing contextual information within the SmartMote – matching static and dynamic context factors to filter the RUM

In the following we will elaborate this idea based on the example of the SmartMote localized within the *SmartFactory*^{KL}. As mentioned above the RUM comprises the specification of interaction zones, a concept initially developed for interaction in working environments [13]. According to the zone in which the user is currently located in, a different use model will be instantiated.

Figure 2 shows an overview of the context concept of the SmartMote. In the concrete scenario the indoor positioning system (UbiSense) as well as the RFID floor grid of the *SmartFactory*^{KL} will be used to determine the exact position of the user. Locations of the tags and geometric dimensions of the interaction zones are defined in the RUM. After receiving data from the positioning system and after contact to a RFID tag was established this information will be added to the RUM. This will result in firing a rule inside the reasoner component: given the information about the position of the user the SmartMote is able to match it to the dimensions of the single interaction zones as described in the RUM. In case a change of the active interaction zones was detected, a filter set can be generated in order to adjust the RUM, which will be interpreted and the displayed interface will be adapted accordingly to only show the functionality that was defined for the particular interaction zone. [1]

At this point the reasoner is only able to adjust the RUM in accordance to the contextual information derived from sensorial input. So far it was not possible to determine if the displayed user interface is still sufficient for the user to achieve her goals, which we will address with this contribution.

4 State of the Art

Before we elaborate our approach to check adapted user tasks, we want to give an overview about previous work on which our approach is built on. On the one hand this work involves task aspects as well as quality facets to check and modify task models. Therefore, in the following we will first briefly discuss task models before we introduce conceptual work on graceful degradation which is usually used in the domain of embedded systems.

4.1 User Task Models

Tasks that are required to be accomplished by the user in order to achieve her goal can be described in a hierarchical manner. Often this supports the stepwise refinement of the goal into abstract tasks down to atomic actions that will be executed during the interaction with the user interface. One of the earliest approaches to describe user tasks is the ConcurrTaskTrees (CTT) notation [8, 10]. Based on the ISO standard LOTOS (Language Of Temporal Ordering Specification) it leverages eight temporal operators to set the single tasks into relation with each other. CTT has constantly been improved but it shares one major drawback with other task models: scalability. Larger models can be created but confuse maintainers which can result in mistakes. Another disadvantage is the degree of refinement of basic tasks because the concrete type and nature of interaction is left open [9].

Based on the Useware Markup Language (useML), the RUM also structures tasks in a hierarchical manner [5]. An advantage of the RUM is the final refinement step towards leaf nodes in the graph structure – the elementary use objects (eUO). These describe the concrete atomic actions which are executed by the user. This makes the interpretation during runtime much easier.

4.2 Graceful Degradation

Graceful degradation is used in the domain of embedded systems to avoid failures in safety critical systems. The idea behind this concept is that a system consisting of diverse components is able to react to malfunctions of single devices and will compensate for the loss of e.g. actuators or sensors. Usually the system will be set into a different state in which it doesn't offer the full functionality anymore but it still is running the basic functionality. Graceful degradation describes the "smooth change of some distinct system feature to a lower state as response to errors" [11].

An example out of the automotive domain is the vitality of a car remaining functional even if subcomponents are not available anymore. In case of a failure of the brake booster the system will switch into a state were the car will only accelerate to a lower speed (than the vehicle's possible maximum). This concept can be transferred to the context of graphical user interfaces, where a failure within the 3D component could result in less graphical effects while still offering the basic functionality [11].

In model-based user interface development graceful degradation has been integrated in order to support the portability [6]. Since the user interface descriptions are available on different levels of abstraction, the portability between different platforms can be ensured much easier than in traditional development processes. Nevertheless, various platforms also provide their individual attributes and are e.g. restricted in terms of display size or resolution, which results in the natural question how to display the same content on a different resolution even if the modality remains the same. For example if the size of an interaction object is restricted graphical user interfaces can be implemented with different widgets that provide the same functionality (but with a loss of comfort, precision, etc.). In case the display space is limited the user interface can offer two buttons for selecting a value within a range (increase and decrease), but if the display space is more ample this function can be displayed as a slider. In the latter case the user can not only specify the value but she is also faster in giving her input by estimating the value on the scale before specifying the precise value. Thus, the concept of graceful degradation was used in this context by Florins [6] to address this problem in model-based user interface development by developing an algorithm, deriving a user interface according to certain platform constraints using a set of graceful degradation rules.

This inspired our work to apply the idea of graceful degradation directly to task models. Because dynamic aspects of ambient intelligent production environments will first have an effect on the tasks the user is allowed to perform it is a consequence to determine the new task sets and to exclude tasks which e.g. will lead into dead locks.

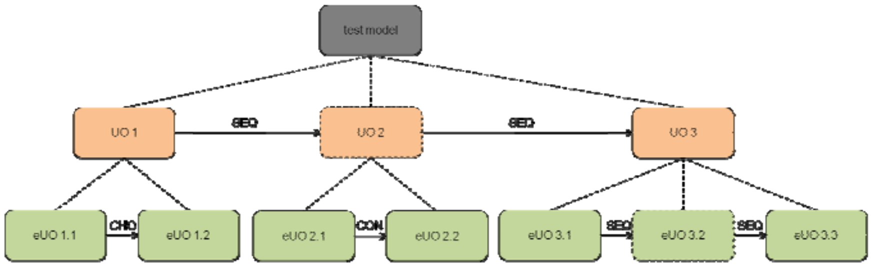

Fig. 3. A sample task model as contained by an RUM

5 Checking for Task Fulfillment by Graceful Degradation

Depending on the derived context, the resulting task model needs to be adapted accordingly. In the following we will introduce our approach based on a dialog model analysis in order to check for task fulfillment. First the underlying dialog model – which is generated out of the task model – will be introduced briefly. This serves as the input for the graceful degradation algorithm checking for task fulfillment and deriving an adapted task model.

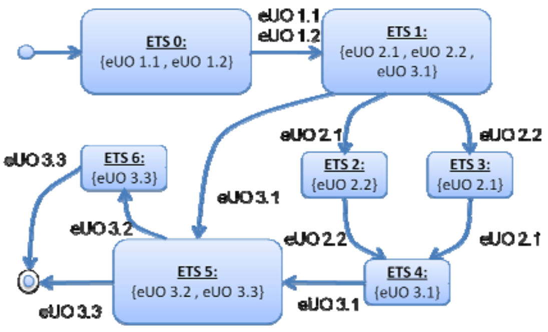

Fig. 4. Dialog model generated out of the task model shown in Figure 3

5.1 Task and Dialog Models

Based on a user task analysis, task models have been proven to be an effective way to formalize tasks that can be conducted by a user on a particular user interface in a hierarchical refined manner. User interfaces consist of dialogs which lead to the integration of a dialog model within our generation process to effectively generate user interfaces [7]. Starting with a task model as depicted in Figure 3, we automatically derive a dialog model in a two-phase process. Staring with a binary priority tree of the task model all possible execution orders need to be stored into a deterministic finite automaton. Enabled task sets (ETS) will be extracted from this binary priority tree by stepwise execution (after which the tree will be updated again) and determining the next ETS. Result will be an automaton (state machine) as required.

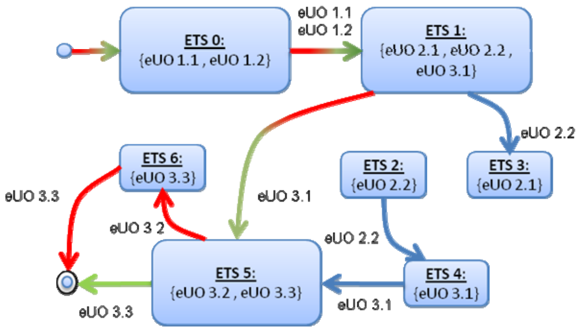

Fig. 5. Dialog model after eUO 2.1 becomes unavailable, removed transitions and marked transitions leading to the accept state

5.2 Task Fulfillment

After the dialog model has been generated it is possible to check if a user can still achieve her goal while using the adapted user interface (generated based on the new model). Figure 4 shows the resulting dialog graph of the task model shown in Figure 3.

Now, in case a service – and the corresponding elementary use object – becomes unavailable, all execution sequences within the model that are in any relation to the unavailable objects will be deleted.

In the given example the elementary use object 2.1 becomes unavailable. Therefore all transitions that are associated to the object will be removed in the automaton as depicted in Figure 5.

Checking all possible paths within this graph than can lead to the accept state the algorithm can determine if this state is reachable. As shown in figure 5, the algorithm will find two possible paths within this model that fulfill this requirement (red and green paths) and as a result declare the users goal as achievable. Four valid execution orders can be found in the example:

- eUO 1.1, eUO 3.1, eUO 3.3
- eUO 1.1, eUO 3.1, eUO 3.2, eUO 3.3
- eUO 1.2, eUO 3.1, eUO 3.3
- eUO 1.2, eUO 3.1, eUO 3.2, eUO 3.3

This is a necessary precondition in order to continue processing with the adaptation because if this check would be omitted the resulting user interface could be – with respect to its functionality – not useable anymore and therefore directly lead to operating errors.

All the elements of the initial task model will be set into the state *not reachable* to be processed by the graceful degradation approach. Starting with the leaf nodes – always elementary use objects – all these elements which are included in the valid execution orders determined from the dialog model analysis will be set to the state *reachable*. In case an object only contains children that are not reachable it will remain in its state. This algorithm will proceed recursively until the root node is reached (see Figure 6).

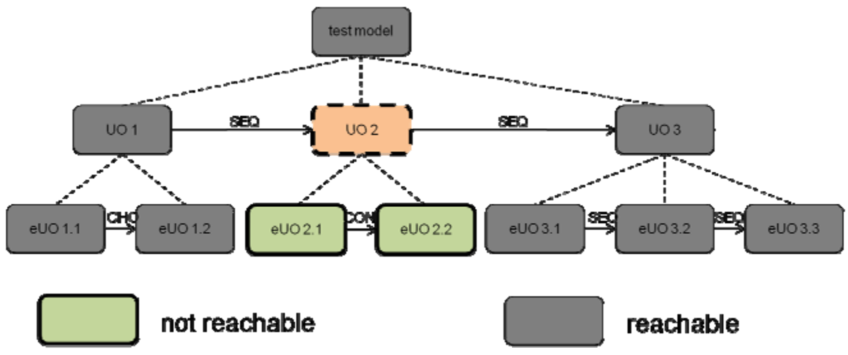

Fig. 6. Task model – marked according to reachability of tasks

Beginning with the root node, all the objects that are marked as not reachable will be removed from the task tree. After this phase is finished the adapted dialog model can be generated which serves as the input for the generated user interface.

5.3 Threats to Validity

First simulations have shown the feasibility of this approach, but also indicated some drawbacks.

Simulated based on models of the *SmartFactory*$^{\mathbf{KL}}$, this approach works quite well (given the fact that it represents a small to medium size production environment), but using larger models will result in a lowered performance and if those models exceed a certain complexity the user interface will not be useable during run-time anymore.

Furthermore, processes within the *SmartFactory*$^{\mathbf{KL}}$ are modeled in a redundant way. This means in case of a malfunction of a pump there are other pumps to compensate its function. Therefore, these redundancies are also reflected within the task model according to the device configurations. Relating to the approach, the user has different ways to reach her goal and in case of an adaptation this can still be possible. If there are no redundancies, the user interface cannot be adapted but also the production environment will not be functional any more in our scenario.

6 Summary and Future Work

Ambient production environments will offer single holistic interaction concepts which offer interaction with all the deployed field devices and therefore will be able to provide a homogeneous user experience. One sample universal control device is the SmartMote which serves as a demonstrator and application scenario for our research.

In this paper we presented one step within our model-based user interface generation approach that is capable of adapting the user interfaces to contextual circumstances. Relying in task models describing the users' interaction with the interface, previous approaches were not able to determine the quality in terms of reachability of the users' goals of the resulting adapted user interface. We elaborated our approach based on this specific tool chain to leverage the determination of valid execution orders after a contextual chance. Based on the detected execution orders the task model can be adapted accordingly, avoiding tasks that cannot be executed by the user any more. Thus, also the resulting user interface does not offer any user interaction or dialogs which cannot be reached by the user or will lead to dead locks within the users' task execution process.

As mentioned above we successful simulated this approach with various models and adaptations based on our living lab while also indicating drawbacks. We therefore need to conduct further investigations on the performance of this algorithm and we are simultaneously developing additional algorithms against we will test the presented approach.

References

1. Bizik, K.: Context Modeling in the Domain of Ambient Intelligent Production Environments, Diploma Thesis, University of Kaiserslautern, Germany (2010)
2. Breiner, K., et al.: SmartMote: Increasing the Flexibility of Run-Time Generated User Interfaces by Using UIML. In: Proceedings of the 2nd International Workshop on Visual Formalisms for Patterns (VFfP 2010), located at the IEEE Symposium on Visual Languages and Human-Centric Computing 2010, Madrid, Spain (2010)

3. Breiner, K., Gauckler, V., Seissler, M., Meixner, G.: Evaluation of user interface adaptation strategies in the process of model-driven user interface development. In: Proceedings of the 5th International Workshop on Model-Driven Development of Advanced User Interfaces (MDDAUI-2010), located at CHI 2010, CEUR-Proceedings, Atlanta, GA, United States, April 10 (2010)
4. Breiner, K., Görlich, D., Maschino, O., Meixner G.: Towards automatically interfacing application services integrated in a automated model-based user interface generation process. In: Proceedings of the Workshop on Model-Driven Development of Advanced User Interfaces (MDDAUI 2009), 14th International Conference on Intelligent User Interfaces (IUI 2009), Sanibel Island, Florida (2009)
5. Breiner, K., Görlich, D., Maschino, O., Meixner, G., Zuehlke, D.: Run-time adaptation of an universal user interface. In: Proceedings of the 13th International Conference on Human-Computer Interaction (HCII 2009), San Diego, CA (2009)
6. Florins, M.: Graceful Degradation: A Method for Designing Multiplatform Graphical User Interfaces. PhD Thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium (2006)
7. Meixner, G.: Model-based Useware Engineering. In: Proc. of the W3C Workshop on Future Standards for Model-Based User Interfaces (MBUI 2010), Rome, Italy (2010)
8. Paternò, F.: ConcurTaskTrees: An Engineered Notation for Task Models. In: Diaper, D., Stanton, N. (eds.) The Handbook of Task Analysis for Human-Computer Interaction. Lawrence Erlbaum Associates, Mahwah (2003)
9. Paternò, F.: Tools for Task Modelling: Where we are, Where we are headed. In: Pribeanu, C., Vanderdonckt, J. (eds.) Proc. of the First International Workshop on Task Models and Diagrams for User Interface Design (TAMODIA 2002), pp. 10–17. INFOREC Publishing House, Bucharest (2002)
10. Paternò, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: Proc. of the 6th IFIP International Conference on Human-Computer Interaction (1997)
11. Saridakis, T.: Design Pattern for Graceful Degradation. In: Noble, J., Johnson, R. (eds.) Transactions on Pattern Languages of Programming I. LNCS, vol. 5770, pp. 67–93. Springer, Heidelberg (2009)
12. Strang, T., Linhoff-Popien, C.: A Context Modeling Survey. Workshop on Advanced Context Modeling, Reasoning and Management. In: 6th International Conference on Ubiquitous Computing (UbiComp 2004), Nottingham, England (2004)
13. Streitz, N.A., Röcker, C., Prante, T., Stenzel, R., van Alphen, D.: Situated Interaction with Ambient Information: Facilitating Awareness and Communication in Ubiquitous Work Environments. In: 10th International Conference on Human-Computer-Interaction (HCII 2003), Crete, Greece (2003)
14. Wurdel, M., Forbrig, P., Radhakrishnan, T., Sinnig, D.: Patterns for Task- and Dialog-Modeling. In: 12th International Conference on Human-Computer Interaction (HCII 2007), Beijing, P.R. China (2007)
15. Zühlke, D.: SmartFactory - A Vision becomes Reality. Keynote Paper of the 13th IFAC Symposium on Information Control Problems in Manufacturing, INCOM 2009 (2009)