Run-Time Adaptation of a Universal User Interface for Ambient Intelligent Production Environments

Kai Breiner[1], Daniel Görlich[2], Oliver Maschino[1], Gerrit Meixner[3], and Detlef Zühlke[2]

[1] Software Engineering Research Group, University of Kaiserslautern, 67663 Kaiserslautern, Germany
{Breiner,Maschino}@cs.uni-kl.de
[2] *SmartFactory*KL, P.O. Box 3049, 67653 Kaiserslautern, Germany
{Daniel.Goerlich,Detlef.Zuehlke}@DFKI.de
[3] German Research Center for Artificial Intelligence (DFKI), 67663 Kaiserslautern, Germany
Gerrit.Meixner@DFKI.de

Abstract. The *SmartFactory*KL is an arbitrarily modifiable and expandable (flexible) intelligent production environment, connecting components from multiple manufacturers (networked), enabling its components to perform context-related tasks autonomously (self-organizing), and emphasizing user-friendliness (user-oriented). This paper presents the results of a research project focusing on the run-time generation and adaptation of a universal task-oriented user interface for such intelligent production environments. It employs a Room-based Use Model (RUM) developed in the context of a continuing research project series on universal remote control devices for intelligent production environments. The *SmartFactory*KL is the first ambient intelligent production environment for demonstration and development purposes worldwide. After three years of research, a first prototype has been finished that allows for controlling the production line using a single remote user interface able to adapt to varying remote devices according to the actual context of use, in a complex, model-based approach.

Keywords: MBUID, Model driven development, generating user interfaces, modeling, adaptable user interfaces.

1 Introduction

The ongoing technological development of microelectronics and communication technology is leading to more pervasive communication between single devices or entire pervasive networks of intelligent devices (smart phone, PDA, Netbook, etc.). Furthermore, distributed computing power continues to increase – also for industrial devices and components. Especially industrial devices and applications can take advantage of modern smart technologies, e.g. based on ad-hoc networks, dynamic system collaboration, and context-adaptive human-machine interaction systems. The

J.A. Jacko (Ed.): Human-Computer Interaction, Part IV, HCII 2009, LNCS 5613, pp. 663–672, 2009.
© Springer-Verlag Berlin Heidelberg 2009

vision of Mark Weiser [12] concerning ubiquitous computing – also in production environments – is becoming a reality.

Besides the many different benefits offered by smart technologies, there are also drawbacks. One main drawback is the fact that the number and complexity of technical devices, their user interfaces, and their usage situations in industrial production environments are constantly growing. In today's production environments, technical devices often stem from multiple vendors with different user interfaces differing in complexity, look&feel, and interaction styles. Such highly complex and networked technical devices or systems can provide any information at any time and in any place. This advantage can turn out to be a disadvantage when information is not presented properly according to the users' needs. This leads to problems, especially concerning the usability of the user interface. The level of acceptance of a user interface largely depends on its ease and convenience of use. A user can work with a technical device more efficiently if the user interface is tailored to the users' needs, on the one hand, and to their abilities on the other hand. Therefore, providing information in a context- and location-sensitive manner (depending on user, situation, machine, environmental conditions, etc.) has to be ensured.

To reduce the usage complexity of user interfaces and improve their usability, one of our goals is to adequately support users in performing their tasks by interacting with a user interface. Therefore, the particular user interface has to be adaptable to different usage situations – definable, for example, by user, task, interaction device, and device functionality. The increasing complexity due to technological development will be reduced by using a model-based approach for the generation of user interfaces [9]. The core model of a model-based approach focusing on user-centered development is often the task model of a user interface. A task model describes the tasks a user wants to perform in a system. One comprehensive task model is the *Use Model*, which integrates detailed information about the tasks, e.g., temporal relationships, conditions, or task types [7]. The Use Model is formalized through the XML-based Useware Markup Language (useML). For describing *Use Models* in ubiquitous environments, the Use Model needs to be extended to include the integration of spatial information, which leads to the *Room-based Use Model*. First evaluation results have been obtained in the *SmartFactory*$^{\mathbf{KL}}$**,** our testbed for future production environments, which is located in Kaiserslautern, Germany.

The remainder of this paper is structured as follows. Section 2 describes the Room-based Use Model on the basis of an enhanced version of useML as well as the function model. Section 3 introduces the model-driven generation process, the interpreter, the adaptation mechanisms, and the first prototype developed. In section 4, we conclude and provide an outlook to the future.

2 The Room-Based Use Model (RUM)

The Room-based Use Model (RUM) is a partial model focusing on the tasks of users and the way they fulfill tasks using multiple devices in complex, highly instrumented environments. In the following, we will describe the enhancement of the original useML and the structure of the function model, which is necessary for the automatic generation process that we will elaborate later.

2.1 Enhancing useML

useML was originally introduced by Reuther [11] (see Fig. 1) to formalize several user groups' task models into a single model. The semantic of the first version of useML was later enhanced substantially [7]. Such a model stores all potential variations of all user groups' approaches to achieving desired goals. This model can then be instantiated at any time to automatically adapt the respective device's user interface to perfectly fit the current user's tasks and needs.

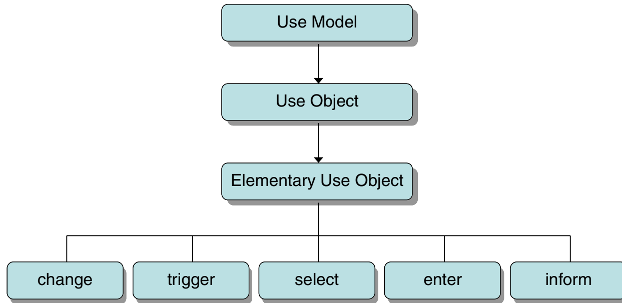

Fig. 1. Use Model structure according to Reuther. [11]

useML was restricted to static user interfaces and to single devices or device families only. It has been extended by a hierarchical structure of – logical (organizational) or physical – rooms containing device compounds that themselves can comprise other device compounds or devices (see Fig. 2.). Thereby, whole business processes can be represented – from a human-machine interaction perspective – in an RUM.

While the RUM provides merely structural elements to specify spatial and device hierarchies, it further allows adding, for example, device profiles, coordinates, and interaction zones. It also provides means for modeling interactions between devices and for defining common Use Models for groups of devices, among other things. In addition to the hierarchical task structure common in task modeling languages (see [8]), the RUM also comprises modeling tools common in software engineering (activity diagrams) and provides support for the application of usability patterns. Still, complex tasks can be refined into less complex and finally elementary tasks (here: elementary use objects) in the classical, hierarchical way.

RUMs can be extended by additional formal elements and sub-structures. When needed, they are supplemented with user models, usage situation models, or other (semi-)formal context representations. An RUM representing the *SmartFactory*$^{\mathbf{KL}}$ was complemented by a function model linking user tasks with data sets of the wireless communication protocols of the *SmartFactory*$^{\mathbf{KL}}$ development and demonstration facility. By using a wireless, mobile interaction device, we were able to automatically generate fully functional user interfaces. The used function model will be presented in detail in the next subsection.

However, since [6] has shown that a model of human-machine interactions must consist of, at least, a task, a dialog, and a presentation model, we needed a mapping

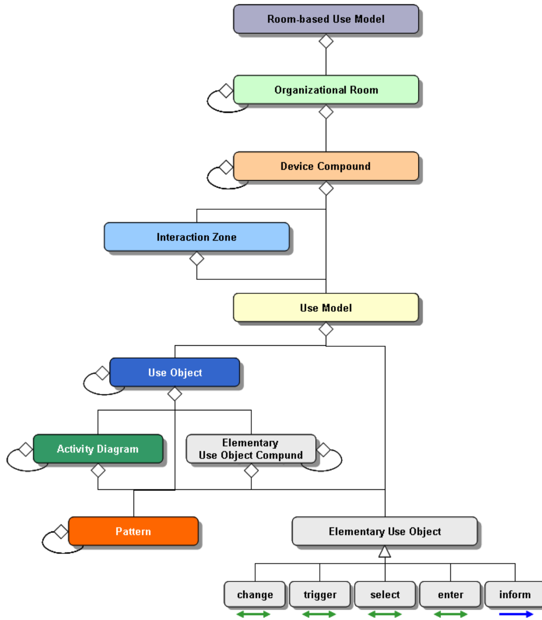

Fig. 2. Integrated Room-based Use Model, containing contextual information about the entire environment, as well as all interactional information about the tasks to be performed by users. [3,4,5]

between tasks and user interface objects (see Section 3) for automated user interface generation and adaptation at run-time. Section 3 will show the feasibility of our approach of combining these models.

2.2 Function Model

The central idea of the function model is to create a linkage between a user interface and the application logic based merely on a given RUM. One major challenge we explored in our previous work was to automatically interface the application services while generating the user interface [1]. For this, we formalized *what* is communicated between the interaction device and the device to be controlled and *how*. This extension makes it possible to bind the application logic to the individual graphical elements with an UI generator in a completely automated way.

These models were elicited on the basis of the PROFIBUS implementation [10] used in the demonstration environment, as in the case of many current production environments. Therefore, it is important to mention that this kind of communication is bi-directional: Once the connection to the target device is established, communication frames can be exchanged cyclically.

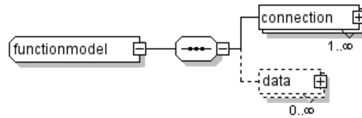

Fig. 3. A device compound node provides a function model consisting of connection information as well as the structure of the data to be transferred

With respect to the structure of the RUM (see Fig. 2.), we created extensions on two different levels of abstraction. First, in order to establish communication between two devices, we needed information about the host to be addressed and the communication channel – this information is device-specific. Second, we needed to know how the content of the communication has to be structured in order to be understood by its receiver.

Due to the fact that in our application domain, the type and channel of communication can vary depending on the type and manufacturer of the device, we attach this specific information to every *device compound* node.

As shown in Fig. 3, the function model consists of the nodes *connection* and *data*. The structure of this model was elicited from several sample projects and implemented with respect to the *uniform resource identifier* (URI) standard [2]. Consisting of *scheme*, *host*, *data-reference* (data-structure, see below), *device number*, *device type*, and *priority*, this information is sufficient for an interpreter to establish a communication channel to this particular device.

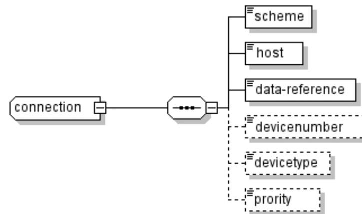

Fig. 4. Connection is a description of the structure of information needed to establish the connection to the desired device

Additionally, it is important to know how to communicate with the respective target device and, therefore, how the transferred data needs to be structured in order to be understood by this device. The PROFIBUS protocol stipulates that communication between devices is, by definition, message-based. Therefore, the content of these messages is embedded into a clearly defined structure, which depends on what kind of information is to be communicated.

Fig. 5 shows that the *data* node consists of distinct structures for incoming and outgoing datasets. Analogously to the protocol, one dataset is composed of a *position* (of the data within the sent/received frame), a *length* (of the information within this frame), a (unique) *identifier*, and a defined *data format*. Additional, but not compulsory, information might be the measurement *unit* (e.g., gallon, liter, Celsius, Fahrenheit), the *conversion factor* (if the data needs to be post-processed),the min/max (possible) range, *significant digits*, and a *status message*.

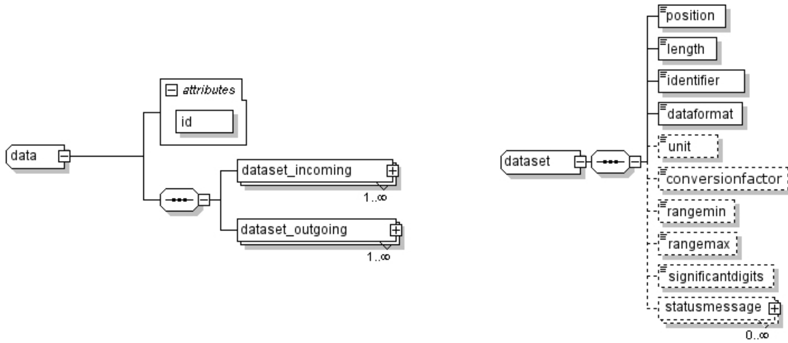

Fig. 5. According to the direction of the communication, the *data* node distinguishes between outgoing datasets and incoming datasets. A dataset consists of several necessary properties.

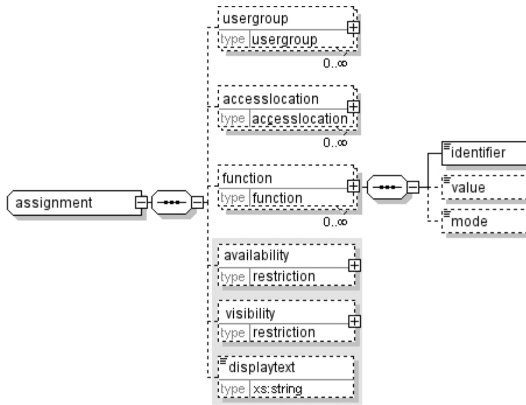

Fig. 6. Each instance of an elementary use object possesses an *assignment* node containing a link to a specific function

Using this structure, an interpreter will be able to encode the information received from a user's interaction in a way the device can understand and vice versa.

The RUM already supports extensions according to assignments on the level of elementary use objects. We equipped each assignment node with the ability to attach a function node pointing to a particular dataset structure, by using the unique identifier (see Fig. 6).

Hence, if an elementary use object is activated by the user, the interpreter knows how to encode the given data according to the dataset structure. Since data structures are assigned to a particular device compound that provides a connection node, the interpreter can send this encoded information.

3 Generation and Adaptation of the User Interface

As a result, the RUM contains information about the user interaction with all devices of the environment as well as information about how to interact with the

corresponding application services, if triggered by the user – which is sufficient for automatically generating a functional user interface. To demonstrate the feasibility of this model-driven approach, we set up an interpretation process and implemented a basic prototype generator.

Prior to this, we formalized our demonstration environment – the *SmartFactory*$^{\mathbf{KL}}$ – as an RUM. From this model, we derived instances that comprise certain subsets of the demonstration environments or certain user roles such as a guest user role restricted to read-only access to the modeled devices.

In the following sections, we will elaborate the process/interpreter, the prototype in the demonstration environment, and our idea of an adaptation mechanism at run-time supported by this model.

3.1 The Generation Process and Interpreter

The aim of this process is to visualize the modeled environment in a single user interface, which will provide control over devices present in the environment. In case of the prototype, we use a simplified presentation model in which the available device compounds and devices will be displayed as a slim navigation structure in a canonical mapping and the use objects will group the functions of the underlying elementary use objects. To activate a certain device (and therefore a certain Use Model), the user selects the respective device in the navigation bar.

Concerning the interpretation of the selected Use Model, let W be the set of available user interface widgets (such as labels, buttons, text fields, etc.) the interpreter will be able to use in order to compose the final user interface. On the other hand, let E_{UO} be the set of elementary use objects and elementary use objects in compound configurations. The function $m{:}E_{UO} \rightarrow W$ describes the relation of how the interaction objects are mapped onto real user interface widgets that are displayed and can be manipulated by the user. This function needs to be formalized in the implementation of the interpreter; in the case of our prototype, it has to be hard-coded. The interpreter will map all (elementary) use objects to a dedicated visual use object (graphical widget).

In addition, the interpreter should not only be able to control devices that are present, but also be adaptive – e.g., hiding the user interfaces of devices that are described in the RUM, but are not really present in the environment.

3.2 Adaptation

According to the RUM, let DM be the set of devices defined in this model – the devices (device types) the interpreter is able to cope with. Let DP be the set of devices present in the environment to be addressed. Available devices are found by scanning for Bluetooth devices, and by querying known Programmable Logic Controllers. Then, the interpreter has to create the intersection between these sets to provide a functional user interface. Hence, $DD := DM \cap DP$ is the set of devices that can actually be controlled with the generated user interface and consequently will be provided by the interpreter.

At this moment, we have an adaptation mechanism for providing a usable user interface that is a direct result of the generation process. Violating this basic type of

adaptation might result in usage-related errors. Assuming $DM/DP \neq \emptyset$ as well as $DP/DM \neq \emptyset$, if only *DM* were to be displayed, the user interface would provide functionality for devices that are not present, and if only *DP* were to be displayed, the interpreter would have to generate user interfaces without knowing how to interface the application logic.

Another adaptation feature supported by the RUM is adaptation according to the role of the user. Analogously to the function model, the user model provides user restriction definitions on the level of the Use Models, attached to each device compound. These definitions result in a restricted view on the Use Model and therefore on the user interface to be displayed. In our scenario, we currently assume that a certain role applies to the complete usage lifecycle of the generated user interface. Thus, $DD' = view(DD, user_{role})$ is the restricted view on the user interface to be displayed according to the actual user role. According to the mapping function defined earlier, the final user interface is the result of the function $m(DD')$.

In the next iteration of our prototype, which is described below, we will integrate more and different implementations of these adaptation mechanisms in order to evaluate them. Due to the fact that this is still research in progress, we have only shown the feasibility so far. But it will be very interesting, of course, to evaluate the influence of different adaptation mechanism on the user experience.

3.3 The Prototype

The prototype was implemented in Java and the generator composes the user interface using elements of the swing library. Xmlbeans was used to create a simple interface to the XML structure of the source files. The wireless communication infrastructure in the *SmartFactory*^{KL} mainly relies on Bluetooth connections, so it was necessary to include a Bluetooth stack in the implementation, which was BlueCove. According to the function model, we included Bluetooth-specific communication information in the connection node, enabling the generator to integrate Bluetooth calls in the action events of the interaction widgets.

Fig. 7. *SmartFactory*^{KL} – the comprehensive user interface running on the PaceBlade touch screen (foreground) is used to steer devices of the production environment (background)

The software runs on a PaceBlade Slimbook P110 TabletPC. It permanently queries the RUM file for changes, in order to let the generator adapt the user interfaces immediately whenever the model is being changed. Fig. 7 shows the generated user interface in the demonstrator environment.

4 Conclusion and Future Work

In this paper, we elaborated the extension of an existing model-based approach to meet the requirements of user interface generation and adaptation in intelligent production environments at run-time. The newly developed Room-based Use Model (RUM) is capable of providing a description of the entire environment, comprising all devices present as well as their user task models. Additionally, we integrated a generic function model, which contains information about the type and way of communication with the application logic, if triggered by a user event. In order to deal with highly dynamic environments, adaptive user interfaces are necessary in order to prevent usage errors. These models were the basis of an automatic model-driven user interface generation process. In our sample production environment, the *SmartFactory*$^{\mathbf{KL}}$, we showed that the information provided by these models is sufficient for creating a functional user interface at run-time. We were also able to integrate basic mechanisms to ensure the adaptability of the user interface to the environmental configuration as well as to the user – which means that the user interface always reflects the configuration of the devices present.

A next step will be to conduct several evaluations using this first prototype. Since the intention was to support the user in interacting with a heterogeneous set of devices (a range of UIs from different vendors providing different user experiences, located in various places, etc.) and to reduce human errors while improving the users' workflow, one goal would be to measure the effectiveness of interacting with our prototype compared to the previous situation of distributed heterogeneous user interfaces. It would be interesting to evaluate how easily users can handle adaptable user interfaces, even if those reflect the current physical configuration. What is the influence on the user experience? Should the adaptation be performed completely automatically or is it possible to integrate the user into this process, improving the level of acceptance? What is the adequate level of user integration: e.g., adaptation only, information, or mutual adaptation?

Acknowledgments. This work was funded in part by the German Research Foundation (DFG).

References

1. Adam, S., Breiner, K., Mukasa, K., Trapp, M.: Challenges to the Model Driven Generation of User Interfaces at Runtime for Ambient Intelligent Systems. In: Proceedings of the Workshop on Model Driven Software Engineering for Ambient Intelligence Applications, European Conference on Ambient Intelligence, Darmstadt, Germany (2007)
2. Berners-Lee, T., Fielding, R., Masinter, L.: Uniform Resource Identifiers (URI): Generic Syntax. RFC. RFC Editor (1998)

3. Breiner, K., Maschino, O., Görlich, D., Meixner, G.: Towards automatically interfacing application services integrated in an automated model-based user interface generation process. In: Proceedings of the Workshop on Model Driven Development of Advanced User Interfaces, 14th international Conference on intelligent User interfaces IUI 2009, Sanibel Island, Florida, USA (2009)
4. Görlich, D., Breiner, K.: Useware modelling for ambient intelligent production environments. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735. Springer, Heidelberg (2007)
5. Görlich, D., Breiner, K.: Intelligent Task-oriented User Interfaces in Production Environments. In: Proceedings of the Workshop on Model-Driven User-Centric Design & Engineering, 10th IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine-Systems, Seoul, Korea (2007)
6. Luyten, K.: Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development, Ph.D. thesis, Limburgs Universitair Centrum, Transnational University Limburg: School of Information Technology, October 21, Expertise Centre for Digital Media, Diepenbeek, Belgium (2004)
7. Meixner, G., Seissler, M., Nahler, M.: Udit – A Graphical Editor for Task Models. In: Proceedings of the Workshop on Model Driven Development of Advanced User Interfaces, 14th international Conference on intelligent User interfaces IUI 2009, Sanibel Island, Florida, USA (2009)
8. Meixner, G., Görlich, D.: Eine Taxonomie für Aufgabenmodelle. In: Proceedings of Software Engineering (SE 2009), Kaiserslautern, Germany (2009)
9. Myers, B., Hudson, S., Pausch, R.: Past, present, and future of user interface software tools. In: ACM Transactions on Computer-Human Interaction (TOCHI), pp. 3–28. ACM Press, New York (2000)
10. PROFIBUS Protocol, http://www.profibus.com/pb/ (last visited 02.02.09)
11. Reuther, A.: useML – Systematische Entwicklung von Maschinenbediensystemen mit XML (Ph.D. thesis) In: Fortschritt-Berichte pak. vol. 8, University of Kaiserslautern, Germany (2003)
12. Weiser, M.: The computer for the 21st century. Scientific American 265(3), 94–104 (1991)