Towards automatically interfacing application services integrated into an automated, model-based user interface generation process

Kai Breiner
Fraunhofer Institute for Experimental Software Engineering (IESE),
67663 Kaiserslautern, Germany
breiner@cs.uni-kl.de

Oliver Maschino
TU Kaiserslautern, Software Engineering Research Group,
67663 Kaiserslautern, Germany
maschino@cs.uni-kl.de

Daniel Görlich, Gerrit Meixner
German Research Center for Artificial Intelligence (DFKI),
67663 Kaiserslautern, Germany
{Daniel.Goerlich, Gerrit.Meixner}@dfki.de

ABSTRACT
The impact of user interface quality has grown in software system engineering. This importance will grow further with upcoming new paradigms such as Ambient Intelligence or Ubiquitous Computing. These paradigms confront the production industry with a new diversity of usage situations. In previous work, we have shown the adaptation of a task-oriented, model-based Useware engineering process to future paradigms by extending existing models and shifting the development/generation of the user interface (UI) from development time to run-time. While separating the UI design from the application engineering process, the problem of the generated UI interfacing with the corresponding service functions emerged. We propose a solution by integrating the respective linkage information into the function model, which will be introduced in this paper.

1. INTRODUCTION
Modern industry production environments are characterized by a heterogeneous set of technical devices, which consequently provide an also heterogeneous set of interaction devices as well as concepts [8]. Also, using modern communication technology, these devices can be interconnected and therefore share information about the current state of the whole environment. Thus, if this particular information is available at any time and in any place, this could also be a disadvantage, namely, if it is not presented properly in terms of format and structure [4]. Confronted with this diversity of interaction concepts and information, it will be more difficult for users to fulfill their tasks or react in proper time in case of an emergency [8]. For this reason, it is important to also consider information about the usage situation (e.g. users' roles, user position, environmental conditions, etc.). Additionally, the use of (independent) information and interaction structures generates new human-machine-interaction concepts and Useware engineering methods [5].

The goal of our approach is to support the users in performing their tasks as adequately as possible. Basically, the idea to solve the problem of "explosive diversity" was to combine all the user interfaces of nearby devices into a single holistic interaction device with a homogenous interaction concept, which results in improved usability as well as in an optimization of the users' workflows. The benefits and the sufficiency of such universal remote controls have already been proven in the context of intelligent households [6]. Faced with a similar initial situation, universal controls also emerge in the domain of production industry [8]. Because intelligent production environments appear to be highly dynamic, it was not an option to provide a simple, static, universal control device. Analyzing a generic environment will always lead to a formal description (model), which in the following will be the starting point of our UI generation approach as described below. The contribution of our paper will be the seamless integration of given functional interfaces into the completely automated generation process, resulting in a fully functional and usable context-sensitive UI. We have developed this process as an extension to the Useware development process, and we will summarize it in the following section.

2. TOWARDS A MODEL-BASED GENERATION PROCESS
Evidently, a modern user interface's level of acceptance is determined by its ease of use. Furthermore, this also applies to entire software products, because for the user, the UI is the product. [17] In order to improve this property, the Useware development process – developed by the Center for Human-Machine-Interaction and successfully applied in numerous joint ventures and industrial projects [7] – has to be adapted systematically.

The starting point of this process is always the systematic analysis of the user and the respective environment, which is the sole guarantee for the efficient use of the final user interface to be developed. Subsequently, the result of this phase will be formalized during the structural design and the design phase, resulting in a room-based use model (useML, see Figure 1), as described in [2]. On the basis of this model, it was the UI programmers' task to implement the final UI. Automating this step between design and implementation is the research focus of the GaBi project, which aims at achieving this goal by defining a model-based code generation process. One major challenge we explored in our previous work was to automatically interface the application services while generating the user interface [1]. We propose bridging this gap by including all the necessary information in an extended model, as described below.

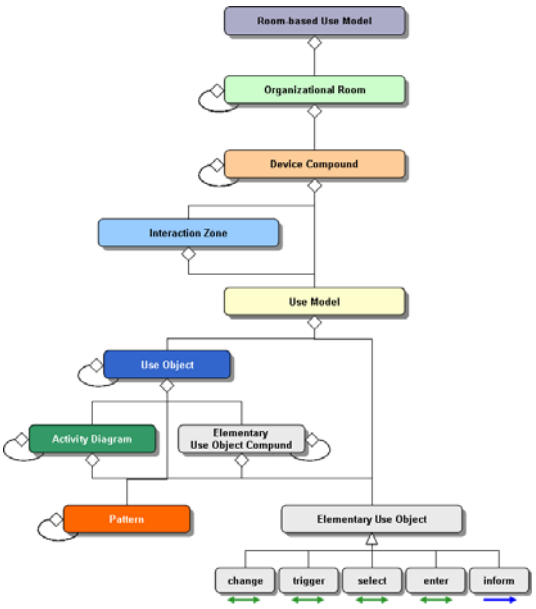

Figure 1. Integrated room-based use model, containing contextual information about the entire environment as well as all interactional information about the tasks to be performed by users. [2]

3. INTERFACING SERVICES

In our scenario, all the devices located in an intelligent production environment to be monitored and controlled already possess a certain predefined set of well-defined service interfaces. The type and communication channel of each service interface strongly depend on the brand or manufacturer of the device. It is also thinkable – as implemented in our demonstration environment – that communication with several devices is encapsulated in or delegated by other communication devices (e.g., Bluetooth DataEagle, PLC, etc.), which is an attempt to homogenize most of the service interfaces to be used. The UI device through which the user should be able to control all these devices has to access all these service interfaces.

Since the intention was to provide a complete UI generation process without manual intervention, the generator needs to know how the respective service interfaces are accessed and how the information needs to be structured when a certain user task is executed. This means that all the information needed to construct the entire UI must be included in the source model – the room-based use model. Like this model, most of the current models provide a detailed description of the human tasks that could be performed, but make no statement about the service functions to be executed.

Existing technologies for describing graphical user interfaces include, among others, the XML-based XML User Interface Language [14] or SwixML [15]. These languages describe a user interface and a specific generator creates the final user interface from their description. This is a semiautomatic process and does not fit our requirements, because with these languages, it is only possible to describe the graphical user interface and there is no information about accessing service interfaces. The result of such approaches is a (compilable) UI source code, containing variation points for manually inserting interfacing code. To meet this challenge, we analyzed the interface definitions and extended the useML model [13] with all the relevant information.

3.1 Challenges

For many reasons, the necessary information is not trivial to provide, because of the communication diversity already mentioned. Only because the application domain of our scenario is restrictive in terms of possible communication ways as well as other factors such as user roles can the risk of the already introduced explosive diversity be handled.

First, common task and domain models, which are usually employed in model-based user interface development (MBUID) processes, assume that all tasks can be canonically mapped to an obvious domain data operation [1]. Yet, in many real environments, service interfaces as well as their manipulating operations are not equivalent to the users' tasks [1]. Therefore, each task can be mapped to one (or a set of) operation(s), which has to be done manually because in most the cases, the underlying semantics are not machine-interpretable.

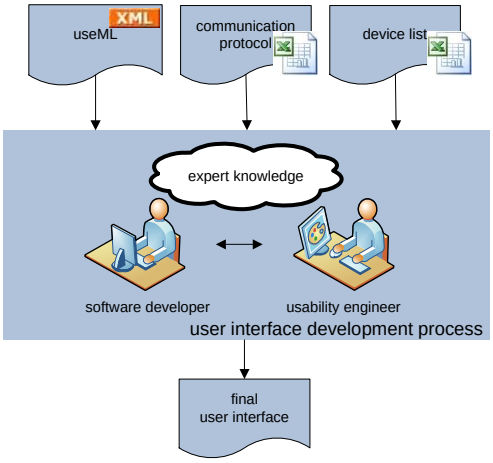

Figure 2. Manual user interface development process on the basis of useML.

Second, the basic idea behind MBUID is to separate domain knowledge from design knowledge. Thus, the idea is that domain experts should be able to create a use model containing all the users' tasks, without having any knowledge about the implementation of the final user interface. The generator includes expert knowledge regarding the user interface design and transforms the model into an efficient user interface. Including information about the communication with an application's interface into the model would imply that the domain expert also needs to have knowledge about some details of the user interface implementation. Hence, the current workflow is visualized in Figure 2, which shows, besides the three major input documents, the roles of both the software developer (technical implementation) and the usability engineer (expert knowledge in design aspects). Among other risks (e.g., human errors), it is also possible that when this process is applied twice using the same documents, it can result in varying output (final user interface) in several ways. This depends on several context factors during the development process, such as the expert knowledge involved in the process.

3.2 Idea – Extending useML

The solution we propose to meet these challenges is to completely automate this process. Consequently, the documents describing the functionality and the interfaces of the devices to be controlled need to be formalized in a machine-interpretable way. Additionally, the obstacle of automatically integrating the necessary code fragments in order to establish communication with the desired devices needs to be conquered. To achieve this goal, we introduce an extension to the sophisticated useML description. In the following, we will call this model the *function model*. Each device compound (see Figure 1) possesses its own function model, because is theoretically possible that each compound needs to be addressed in a unique way. In detail, it consists of two sections:

- Connection – including information about establishing the connection to the device to be controlled
- Data – a basic structure of the content of the communication

One important aspect of this extension is the manner of communication, which is described in the node connection. This model was elicited from sample projects and implemented with the use of the uniform resource identifier (URI) standard [16]. Hence, the general communication information can be stored in a generic way. Therefore, it contains the following information:

- Scheme – define the kind of communication and the needed additional parameters
- Host – the host to be addressed according to the scheme
- Data-Reference – reference to the data structure that the information needs to be encoded with
- Priority – if multiple schemes are available, in order to choose the most adequate one

Optionally, it is possible in our case to add parameters to describe the communication in more detail, which is specific in our sample environment:

- Device-Number – an unique number of the device to be addressed
- Device-Type – the concrete type of the device (e.g. XY-Pump)

Now that a UI generator knows how to communicate with the production environment, it is necessary to encode the transmitted content as well as its semantic. In order to develop the appropriate model, we analyzed sample device descriptions recorded in spreadsheet files that contain the composition of Bluetooth frames. In regular projects, Useware developers used these documents to choose and adjust the widgets of the UI, but also to code the action events and the communication with the environment using the widespread Profibus protocol. This led to a data model attached to each elementary use object. In accordance with the message-based communication in the demonstration environment, the data model provides information for processing incoming frames and for constructing outgoing frames according to the current user input. A data node in our model contains:

- Position – the starting position within the communication frame
- Length – the number of the data blocks used for one data set
- Identifier – of the data set
- DataFormat – basis of the interpretation of the content

Also, there are optional properties of the data model:

- Unit – human-understandable identifier of the data content
- ConversionFactor – if the content needs to be post-processed
- RangeMin/RangeMax – boundary conditions of the value
- SignificantDigits – the number of the significant digits
- StatusMessage – special device-dependent status message encoded in one bitvector

This data model in combination with the communication model forms the integrated function model, which allows for deducting from elementary use objects how certain bits of information need to be transferred in order to execute the desired application logic.

Figure 3. Accessing application services using the Universal Control Device.

4. FEASABILITY STUDY – DEMONSTRATOR

To show the feasibility of our approach and that of the function model, we implemented a basic generator that accepts the adjusted useML specification as input. In general, all elementary use objects are mapped to an object that will be displayed on the screen – the interpretation of task constellation towards widget composition is still ongoing work. The device compound structure of the room-based use model is canonically mapped to a generated simple navigation structure on our sample UI. Thus, a user is able to select a device of the compound and perform the tasks as specified in the useML description. Generating a functional UI was the major purpose of this rudimentary generator. How the generator integrates the communication will be elaborated in the following, where the universal control UI will be considered a product of the generation process.

In general, due to the structure of the extended useML, each elementary use object possesses (if necessary) its own function node, which is a link to a certain data entry in the function model

of its device compound. Thus, if the user interacts with the user interface (triggering a particular elementary use object), we are able to identify the corresponding function, extract the user data from the user interface, and compose the communication frame with the help of the function mode and vice versa.

For the purpose of demonstration, we installed the user interface generator on a Paceblade Slimbook P110 TabletPC [10]. This device possesses a 12.1″ touch screen, which can be used without keyboard or mouse.

We analyzed the communication of our sample environment – the *SmartFactory*^{KL} – and filled in the new models, which are now part of the room-based use model. Figure 3 shows the universal remote device (foreground) in action while controlling devices of the *SmartFactory*^{KL} (background).

5. CONCLUSION AND FUTURE WORK

The result of our study is that it is feasible to provide all information on a certain industrial production environment in order to enable an automatic user interface generator to create a functional user interface.

But there are certainly some basic limitations. We extended a given user interaction model with communication information, because the application domain was clearly restricted. Thus, we did not have to face the problem of combinational explosion (restricted set of types of devices), which always occurs when there are infinite options of combinations between devices, communication channels, etc. On the contrary, in our environment, there is a clearly defined and standardized communication protocol and a predefined set of device types.

The advantage of our approach is that we facilitated the development of a completely automated user interface generator on the basis of only a user interaction model, an environmental description, and the description of the manner of communication. Furthermore, this allows for developing universal control devices that are able to adapt to changing peripheral constellations and always provide an adequate user interface.

Since we now have an automated development process, it is our vision to improve the usability of the generated user interface by including a pattern repository that the generator can make use of. This will lead to user interfaces providing holistic interaction (look'n'feel) for the control of various devices. Beside other effects, we expect to significantly reduce the number of human errors that result from switching between different interaction-concepts. Also, the reaction time in case of critical situations might be reduced, which is vital in production environments.

6. ACKNOWLEDGMENTS

Our work as well as the GaBi project is funded in part by the German Research Foundation (DFG).

7. REFERENCES

[1] Adam, S., Breiner K., Mukasa K. and Trapp, M. 2007. Challenges to the Model Driven Generation of User Interfaces at Runtime for Ambient Intelligent Systems. Workshop: *Model Driven Software Engineering for Ambient Intelligence Applications*, European Conference on Ambient Intelligence, Darmstadt.

[2] Görlich, D. and Breiner, K. 2007. Useware modelling for ambient intelligent production environments. Workshop: *Model-Driven Development of Advanced User Interfaces*, MoDELS 2007, Nashville.

[3] Görlich, D. and Breiner, K. 2007. Intelligent Task-oriented User Interfaces in Production Environments. Workshop: *Model-Driven User-Centric Design & Engineering*, 10th IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine-Systems, Seoul.

[4] Bödcher, A., Mukasa, K. and Zühlke, D. 2005. Capturing Common and Variable Design Aspects for Ubiquitous Computing with MB-UID. In: *Proceedings of the Workshop on Model Driven Development of Advanced User Interfaces*. Montego Bay, Jamaica.

[5] Zuehlke, D. 2004. *Useware-Engineering für technische Systeme*. Springer, Berlin.

[6] Adam S., Breiner K., Mukasa, K. and Trapp, M. 2008. An Apartment-based Metaphor for Intuitive Interaction with Ambient Assisted Living Applications. 22nd European Conference on Human-Computer Interaction HCI2008, Liverpool.

[7] Zühlke, D. and Thiels, N. 2008. Useware engineering: a methodology for the development of user-friendly interfaces, in: Library Hi Tech, Vol. 26, No. 1.

[8] Hofmann, T. and Holzkämper, P. 2008. NEW HMI – Möglichkeiten und Grenzen abstrakt-geographischer Visualisierung in Bereich der Anlagensteuerung. In: Brau, H., Diefenbach, S., Hassenzahl, M., Koller, F., Peissner, M. and Röse, K. (Editors), *Usability Professionals 2008*, pp. 204-208, Fraunhofer IRB Verlag, Sep 2008.

[9] Grund, M. 2006. *Kommunikationstechnologien in der modernen Prozessleittechnik – Mit praktischer Demonstration der dezentralen Parametrierung von Industriegeräten via Bluetooth*. University of Kaiserslautern.

[10] http://www.paceblade.com, last visited 23.09.08.

[11] Bödcher, A. 2007. *Methodische Nutzungskontext-Analyse als Grundlage eines strukturierten USEWARE-Engineering-Prozesses.* Fortschrittberichte pak, Volume 14. University of Kaiserslautern.

[12] Maschino, O. 2008. *A Strategy for Automated Generation of Graphical User Interfaces based on the Useware Markup Language in the Domain of Intelligent Production Environments*. Diploma Thesis, University of Kaiserslautern.

[13] Görlich, D., Thiels, N. and Meixner, G. 2008. Personalized Use Models in Ambient Intelligence Environments. Proc. of the 17th IFAC World Congress, Seoul.

[14] https://developer.mozilla.org/en/XUL, last visited 23.09.08.

[15] http://www.swixml.org, last visited 23.09.08.

[16] Berners-Lee, T., Fielding, R., and Masinter, L. 1998 *Uniform Resource Identifiers (URI): Generic Syntax*. RFC. RFC Editor.

[17] Trapp, M. 2008. *Generating User Interfaces for Ambient Intelligence Systems*. PhD-Thesis, Software Engineering Research Group, University of Kaiserslautern.