Capturing Common and Variable Design Aspects for Ubiquitous Computing with MB-UID

Alexander Boedcher, Kizito Mukasa, Detlef Zuehlke
Center for Human-Machine Interaction
at the German Research Center for Artificial Intelligence
P.O.Box 3049
67653 Kaiserslautern, Germany
Phone: +49 631 205 3570
Fax: +49 631 205 3705
[boedcher|mukasa|zuehlke]@mv.uni-kl.de

ABSTRACT

Developing user interfaces for ubiquitous environments is a challenging task. In such an environment, users can apply different devices to accomplish the same or different tasks. In order to support the users, there should be similarities between user interfaces on these different devices. Hence the user interfaces need to be homogenous. The user interface developer has to make sure that this requirement is fulfilled. There are two approaches; either to design the user interfaces separately or to find a way of defining common aspects once and then address device specific aspects separately. Since the homogeneity is difficult to reach with the first approach – let alone the fact that much effort and time need to be invested – this paper defines a concept for the second approach concentrating on applications in the production automation. Common and variable design aspects are discussed and a solution approach is presented.

Keywords

MB-UID, use model, user-interface model, Useware, useML

1. INTRODUCTION

As one of the results of the rapid technological advancement humans will be surrounded with different devices to help them perform their tasks. This phenomenon of different interconnected devices is referred to as Ubiquitous Computing (in America) or Ambient Intelligence (in Europe). They all define the fact that users will have different, somehow unnoticeable, devices at their disposal for the same or different tasks. These are characterized by the miniaturization and embedding of microelectronics in other objects as well as ubiquity and intelligence [3].

Ubiquitous devices are defined by three interfaces [1]: 1) a (wireless) network interface to ensure access to other devices or data bases. 2) Sensors to gather information from the environment and actors to affect the environment. 3) The user interface to allow humans the cooperation with different devices.

The development of user-oriented interfaces is already a challenging discipline. The interactions with different cross-linked devices will even boost this challenge. The number of available information for the users will increase and the complexity to interact with technical systems will get higher. Platform wide structures for the usage of technical systems have to be developed to support users in handling their tasks with different devices.

Also the number of different devices in ubiquitous environments is mostly not exactly predetermined. Thus the developer is facing higher complexity as well. Developing user interfaces separately for each device will lead to very high development efforts [4] and the risk of implementing different usage structures. Therefore defined development strategies have to be installed.

A model-based approach to define common and variable aspects of different devices is the most promising approach that leads to a ubiquitous environment where:

1) users can accomplish their tasks (on all available devices) in an intuitive way using the same usage structures for similar tasks.

2) developers have a manageable development effort by defining common design aspects once and just add the variable aspects for each device.

The following results concentrate on the domain of production automation that has some domain-specific restrictions to be assumed.

2. ANALYZING COMMON AND VARIABLE DESIGN ASPECTS

Before defining models, it is important to make an analysis of common and variable design aspects. Till now, there has been no clear differentiation of this point. Likewise, the term “platform independent” has not been uniformly used. Some will call a model “platform independent”, when it contains no specific implementation terminologies like “window” or “button” etc. Yet others will refer to models with these terminologies but that can be adapted to different platforms as “platform independent”. Clarification of this point is therefore an important issue, if the model driven approach will come to success. By separating common and variable aspects, it becomes obvious for developers, which parts need to be changed in different models describing the user interface. Figure 1 shows common and variable design aspects for the production automation domain.

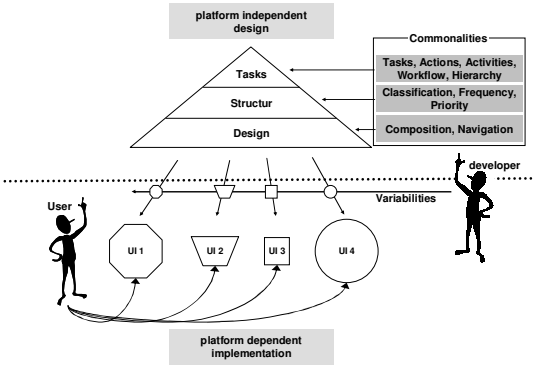

Figure 1. Common and variable design aspects

2.1 Commonalities

The following commonalities were identified for the domain of production automation. It has to be mentioned that these commonalities find their way into the model-driven development process in different process steps shown in section three to six:

Tasks: For a user-oriented approach the consideration of hardware aspects is secondary in a first instance. Tasks, actions, activities and operations describe the complexity and dynamic aspects of users work in a technical surrounding.

Context: The context tasks are fulfilled in is the same on different devices. Context might be user group (worker or technician), usage situation (production or breakdown), surrounding (bureau or factory floor) and others.

Frequency/Priority of usage: Tasks of special importance or security relevant tasks can be marked to assure special treatment within different devices

Navigation control: General aspects of navigation should be established to assign dynamic sequences to different devices.

Layout/Composition: Abstract terms of design can be expressed platform independent. This means e.g. relative positioning aspects.

The consideration of commonalities helps developers to map platform independent aspects to all development models.

2.2 Variabilities

By considering all commonalities there are only little parameters left for defining concrete interaction platforms. These are only briefly addressed by examples to describe their general appearance.

Hardware-specification: The final definition of the hardware platform gives way to the final design of user interfaces. Hardware specification deals with e.g. sizes, interaction-elements, modalities which define fonts, colours, output modalities, etc.

Technical aspects: Besides the user interface technical specifications are made. These include interface definitions or communication protocols.

Localisation: Consideration of different markets brings up aspects of language and representation.

3. THE MODEL-DRIVEN USEWARE[1] DEVELOPMENT PROCESS

A model driven approach requires that models are defined and generated at specific stages in the development process. Therefore there must be a specific flow of development activities. When developing user-interfaces for machines, the Useware development process indicated in Figure 2 is applied. This consists of four overlapping phases accompanied by an iterative evaluation phase. The iteration ensures that the results of each step are accessed not only by the developers but also by the final users [6].

Starting with the **analysis** phase data about user tasks, their mental model, machine details, the working environment as well as the organizational structure is collected. Several data collection methods including interviews, direct observation of workers in the workspace and questionnaires should be applied, since each technique will only give partial information. The results are mainly documented in a preliminary task model. The analyzed data provides the data base for all following development phases.

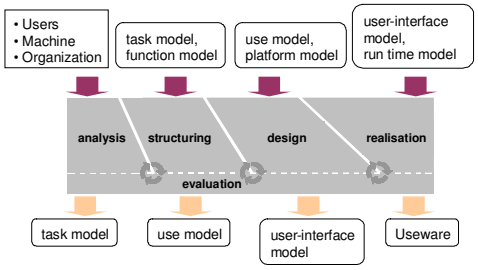

Figure 2. The model-driven Useware development process

Structuring the preliminary task model follows the analysis phase. The previously defined task model and the model of machine functionality are the main input of this phase where only common usage aspects are addressed. This includes defining user groups and their tasks, the usage context or the accessibility of tasks at different devices and locations. It should be mentioned that devices do not play any design role at this stage. They only serve as filters that can be used to decide which tasks are available at which device. The resulting use model can be evaluated in terms of logical grouping, decomposition and others. It means, for example checking whether each task has been placed in the right context and if proper decomposition has been done. The use model is independent of the later implementation platforms.

Once the use model has been defined, user interface **design** can begin. In the first step, further common design aspects (Commonalities) are addressed in the abstract user interface model. The next steps results into a concrete user interface model that refines the abstract model by defining dialog objects and other platform specific aspects (Variabilities). UI-Prototypes can be directly generated from this model and tools can be developed exporting the model into required programming languages.

Hard coding (programming) the GUI and implementing it on the target machine is the task of the last phase; **realization**. The

[1] Since '98 Useware is applied as collective term for all hard- and software components used for operating technical systems [6].

platform also offers hardware capabilities, like for example switches and hard-keys. The resulting Useware can then be evaluated regarding design issues and real time performance.

4. THE USE MODEL

The task model has proven to be a good starting point for user-oriented interface development [4]. Its feasibility to capture user tasks and the way they are performed leads to a focus on the final user during the development process. The preliminary task model is therefore the basis for the use model.

The use model is defined by using useML. This is an XML-based markup language for defining and structuring user tasks for machine users [5]. Its main description elements are the use objects (UO) and the elementary use objects (eUO). While the UOs are logical equivalent to one or more related tasks, the eUOs are the elementary actions. A use object therefore expresses a general goal of one or more tasks. The useML elements and their relationships are indicated in a simplified UML class diagram shown in Figure 3.

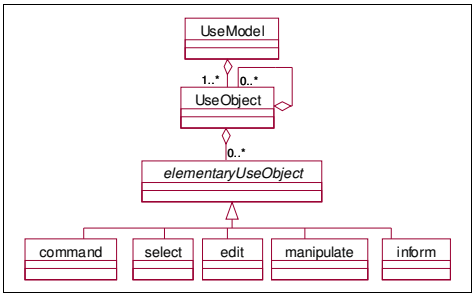

Figure 3. The main elements of useML

Since the meaning of the use model and the use objects is self-explanatory, only details of the elementary use objects will be provided here. As Figure 3 shows, there are five types of eUOs; *select*, *edit*, *manipulate*, *command* and *inform*. These correspond to the actions of the machine user and can fully describe all interaction and information needs of users working with technical systems.

select defines actions where the user can select one or many values from a set of values that already exist in the system. This selection can lead to changing a parameter in the machine control, for example, changing the unit of speed from km/hr to m/s, or to triggering a machine function, e.g., changing the machine operation modus from “automatic” to “manual” by selecting the required modus.

edit involves input of one absolute data value into the machine system. Any available value will be overwritten.

manipulate is basically like edit with the exception that changes are made relative to the existing value. It is therefore possible to increment the speed from 15m/s to 17m/s with a pre-defined incrimination factor. Logically only numeric values can be manipulated.

command implies that the user can directly trigger an action or a machine function resulting into its direct execution.

inform involves the user querying the machine for some information. For example the user would like to know the status of the machine. No further interaction is expected here.

With these few but elementary elements, it is possible to define the use model in a platform independent way as the elements are directly deduced form users tasks and extended by commonalities like classifications, priority, etc. For the schema of the use model and useML refer to [5].

5. THE PLATFORM MODEL

Another model is specified on behalf of the interaction platform. As the name already implies, the model is platform specific. However, it is possible to define a family of platforms that share the same features. A platform is defined in terms of its hardware specifications, its layout and supported dialog objects as well as the available interaction devices. Ergonomic design rules to be observed can also be provided. Hardware specifications include the size and type of the platform and its resolution.

The platform layout defines the partition of the platform into regions that hold logically related objects. For example, there may be region for navigation, for direct function keys and a workspace region for data display and manipulation [7]. The workspace region differs from the other two while it contains dynamic content. It is a main area where the user can view, enter or change data. This can further be partitioned into message and status regions and orientation (see Figure 4).

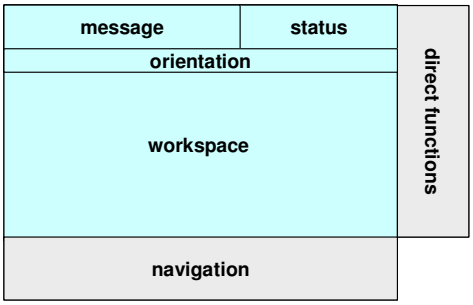

Figure 4. An example of a platform layout

6. THE USER-INTERFACE MODEL

In order to properly address common and variable design aspects, the user interface model is organized into abstract and concrete levels. Here each presentation and interaction is addressed separately, resulting into a presentation model and an interaction model as seen in Figure 5.

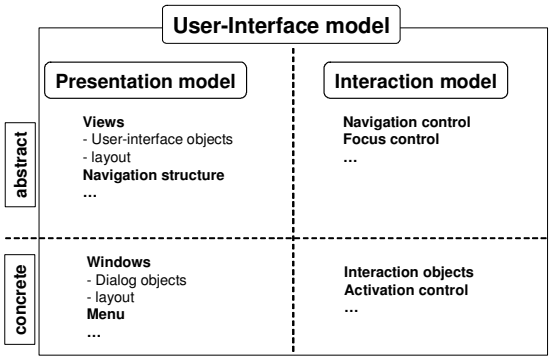

Figure 5. Organization of the user-interface model

6.1 The abstract user interface model

The abstract user interface model uses special elements to address platform independent aspects. At the abstract presentation level, **views** and **user-interface objects** are used as abstractions of screens and dialog objects. A user-interface object can be a container of other objects or a simple one. Simple user-interface objects can be of type **edit**, **select** or **trigger**. This corresponds to input fields where data can be entered or edited, a selection field with constant or variable data and elements for triggering direct actions.

In order to define the layout for a view in a platform independent way, the neighborhood principle is applied. This means that the relative position of each object is defined by specifying its nearest neighbors on its four geographical sides; **east**, **west**, **north** and **south**. Figure 6 demonstrates the application of this principle to define the ordering of two objects in the xml-code of the user-interface model and the resulting user interface.

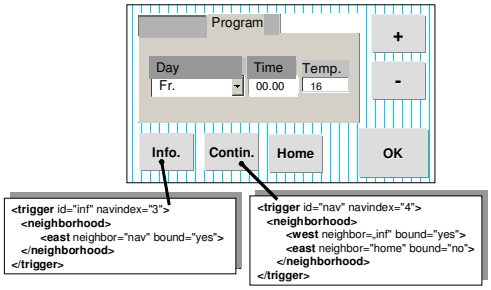

Figure 6. An example of applying the neighborhood principle

The **navigation structure** should also be common to all user-interfaces that are based on the same model. The structure does not only define the number and ordering of the views, but also their topology. The topology can be linear, tree like or network like [2]. Depending on the type used, it can be possible to move from one view directly to a hierarchical neighbor of the same branch or to another branch.

In the interaction part of the user-interface model, **navigation** and **focus** control are defined. While the navigation structure defines the navigation topology, the navigation control specifies how the change from one view to another should take place. The change can either be automatic or it can be triggered by the user. An example of the first case is automatically changing from any view to a view locating the error source when a system error occurs. In the second case, the user can explicitly trigger the change, for example by pressing a button. The change can also be associated with conditions. For example it can be stated that the change from one view to another should only be possible when the user inputs are complete and valid.

Focus control defines the change of sequence of the input focus for the user-interface objects on one view. This can be accomplished by providing a local index for each user-interface object. The index is local since its scope is within a view or a container.

6.2 The concrete user interface model

Having defined the user interface in an abstract way, the next step is to map the user interface onto a concrete platform. Dialog objects that are supported by the goal platform are identified and the values for their attributes are specified. For example, their position on the display as well as their size can now be specified. The decision is made according to the requirements contained in the abstract interface objects. It is important to mention that these are general usage requirements and not design rules/principles. It is left to the platform to find a way of meeting these requirements. Of course ergonomic rules have to be obeyed.

The specification of the platform can be obtained from the platform model. Once the platform is known, the number and location of **interaction objects** can be determined. These are special objects, which may be required to interact with other objects on the user-interface. Basic usages of interaction objects are navigation, focus control or triggering. Depending on the type of platform, some might not be needed. For example no focus control objects might be needed on a touch screen.

After determining the interaction objects, the **activation control** can be defined. This specifies which object or object combination is required to activate an action. This is especially important due to security reasons. For example it can be required that two interaction objects must be pressed at the same time in order to start the machine.

7. Conclusion and future work

This paper has presented an approach for UI-development for ubiquitous environments. It follows the model-based approach where design aspects are captured by using different models. These are products of the model-driven Useware development process. Emphasis has been put on a clear definition of common and variable design aspects, since this is a basis for the distinction between platform independent and platform dependent models. Platform independent aspects have been addressed in the use model and in abstract user-interface model. The concrete user-interface model and the platform model define platform specific issues. Efforts of providing tool support for the development process are underway.

8. REFERENCES

[1] Aarts, E.: Ambient Intelligence – A new user experience. "Philips Ambient Intelligence". Website, http://www.research.philips.com/InformationCenter/Global/FArticleSummary.asp?lNodeId=712, 2002

[2] EN ISO 14915-2: Software ergonomics for multimedia user interfaces- Part 2: Multimedia navigation control, 2003

[3] Mattern, F.: Total vernetzt, Springer, Berlin, Heidelberg, New York, 2003

[4] Paterno, F.: Santoro, C.: One Model, many Interfaces. In: Computer-Aided Design of UI, Kluwer Academic Publishers Dordrecht, Boston, London, 2002

[5] Reuther, A.: useML – Systematic Useware-engineering with XML. Kaiserslautern University of Technology, 2004

[6] Zühlke, D.: Useware-Engineering für technische Systeme, Springer, Berlin, Heidelberg, New York, 2004

[7] VDI/VDE3850-Guideline: User-friendly design of useware for machines, 2000