# A MODEL-BASED APPROACH FOR USEWARE DEVELOPMENT

**Kizito Mukasa, Dirk Ziegeler, Detlef Zuehlke**

*Institute for Production Automation, Center for Human-Machine-Interaction*
*Kaiserslautern University of Technology*
*D-67653 Kaiserslautern, Germany*
*Phone: +49 631 205 3570*
*Fax: +49 631 205 3705*
*Email: [mukasa|ziegeler|zuehlke]@mv.uni-kl.de*

Abstract: The development of human-machine interfaces today is being challenged by the increasing number of interaction devices and multimodality. Developing for multiple devices but yet keeping consistency is unavoidable. Also integrating the future users in the whole development process is important for the final acceptability. Hence the development process is complex. This paper suggests a model-based approach. Aspects of the user interface should be defined in different models of different abstraction levels. This also requires a simple and domain based description language. For this purpose, a XML-based markup language called *useML* is also introduced. useML defines a simple syntax for the description of a platform-independent and task-oriented use model. Platform-specific interfaces and prototypes can be easily derived from the use model.
*Copyright © 2005 IFAC*

Keywords: user interface, prototyping, models, task orientation, objects

## 1. INTRODUCTION

The increasing number of platforms and functionality, the critical requirement to cut off development costs and at the same time increasing the usability of user interfaces requires a systematic approach. A number of solutions have been proposed including developing several versions of the same application for the same platform, using style sheets application and the model-based (MB-UID) approach (Szekely, 1996). The last solution is more comprehensive and has been a subject of research especially from the late 80s. The introduction of XML inspired new life in this field, due to the easy portability of interface descriptions. Nevertheless, proper implementation of MB-UID with XML is still a research issue. Identified problems include; high level of abstraction that makes the UIDLs too general and hence failing to meet domain specific requirements, lack of support tools, difficulties in integration and data exchange between the models, lack of application advice for developers, etc (see also Myers *et al*., 2000). Especially the first three are very critical problems and they are the subject of this paper. A model-based Useware development process is introduced together with an accompanying description language. Useware is a collective word for hardware and software necessary for operating a machine.

## 2. THE USEWARE DEVELOPMENT PROCESS

As highlighted in the introduction, Useware development requires a systematic approach. It consists of a phase based horizontal *macro*-process and a short and repetitive vertical *micro*-process that is embedded in each phase of the *macro*-process (compare Fig. 1). The *micro*-process includes the identification of inputs and outputs, and the activities required to transform the input into the output. As

shown in Fig. 1 the *macro*-process consists of five overlapping phases that are linked to a continuous evaluation block; analysis, structuring, design and realization.

Starting with the **analysis** phase, user tasks, their mental model, machine details, the working environment as well as the organization structure are colleted. Several data collection methods including interviews, direct observation of workers in the workspace and questionnaires should be applied, since any technique will only give partial information about the task (Wilson and Johnson, 1996). The results are documented in a preliminary task model

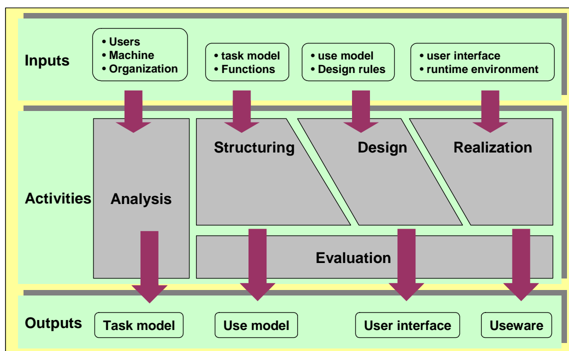


Fig. 1 The Useware development process

**Structuring** the preliminary task model follows the analysis phase. The previously defined existing task model and the machine functional model are the main input of this phase.

First, the tasks are grouped into logical contexts that reflect their usage structure. Examples of contexts in the production automation domain are production, configuration, diagnostic and maintenance. Thus all tasks that are related to the production context, i.e., normal machine operation, are moved to this group.

Then the tasks are hierarchically decomposed into sub-tasks. The decomposition terminates with actions, which can no longer be decomposed. This forms a tree like structure, with actions as leaves. After this decomposition, machine functions and related data that are needed for the internal completion of the task are attached to the corresponding action. User and location restrictions can also be specified, i.e., stating which user is allowed to perform which task and from which location or device.

The resulting use model can be evaluated in terms of logical grouping, decomposition and other restrictions. 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.

The **design** phase should logically follow the structuring phase. But the experience made is that there is some overlapping between the two. Therefore starting the interface design, i.e., making the use model accessible to the user machine user does not wait until structuring is completed. In fact, some "lose" design issues affect structuring. The size of display, for example, can constrain the number of contexts; the less the display, the fewer the number of elements per context group.

As mentioned above, the main goal here is to make the use model accessible. First, the use model is transformed into an abstract user interface model by defining abstract user interface objects (aUIO) for the use objects. Objects for navigation and interaction, which basically have no corresponding use objects, are also defined. The aUIOs are then mapped onto platform dependent concrete user interface objects (cUIO) in a concrete user interface model (see the next sections for more details).

The output of this phase is a prototype of the graphical user interface (GUI) that runs on the development computer. It can be evaluated with respect to layout, navigation and interaction. Any changes can be easily implemented before the GUI code for a target programming language is generated.

Hard coding (programming) the GUI and implementing it on the target machine is the task of the last phase; **realization**. The platform also offers hardware capabilities, like for example switches and hard-keys, which can be linked to their corresponding cUIOs. This setup is the final product; the Useware. In the best case, the skeleton of GUI-code should be generated from the GUI Prototype of the previous phase. This is a long time goal of the author of this paper.

## 3. USER INTERFACE MODELS

Having described the development process, this section is dedicated to user interface models and objects, which are either inputs or intermediate outputs of the development activities.

### *3.1 The preliminary task model*

The preliminary task model gives an overview of the user groups, their main contexts and all tasks together with the information or tools that user requires for accomplishing the task. No distinction is made between tasks that are direct interactions with the machine and those that are not. Fig. 2 shows an example of a preliminary task model. It has not been taken from any existing situation, but is used to demonstrate the concept. Some of the tasks of the operator are *to monitor production* during normal *production* and to *assist the technician* during *maintenance*. Monitoring involves *watching parameters*, *adding raw materials* as well as *reading the manual*. For this purpose, the user needs a *parameter overview*, a *production plan* and a *manual*.

Assisting the technician involves *fetching the tool box*, *holding the maintenance manual* and *cleaning the workspace after maintenance*. The required tools have also been indicated. As it can be seen, this task is not directly related to the user interface. Considering the relevance of a task to the user interface is done in the development of the use model.

| User group | Context | Task/Goal | Action | Information/ Tools |
|---|---|---|---|---|
| Operator | Production | | | |
| | | Monitor production | watch parameter | Parameter overview |
| | | | | |
| | | | add row material | Production plan |
| | | | | |
| | | | read manual | Manual |
| | | | | |
| | Maitanance | | | |
| | | Assist technician | fetch tool box | Tool box |
| | | | | |
| | | | hold maintanance manual | Manual |
| | | | | |
| | | | clean workspace | Brush |

Fig. 2 An example of the preliminary task model

### 3.2 *The use model*

The task model has proven to be a good starting point for user-oriented interface development. This is because it captures user tasks and the way they are performed. Thus, making users the focus of the development. The preliminary task model is therefore the basis for the use model.

The definition of this model is done by using useML. useML is an XML-based user interface description language. It was originally developed for the description of machine Useware (Reuther, 2003). The specialization enables proper addressing of domain specific requirements. The notation has been somehow modified as compared to (Reuther, 2003; Mukasa, 2004) to better fit task modeling. The 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 Fig. 3.

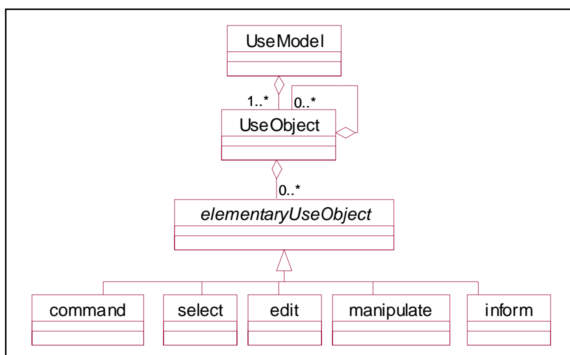


Fig. 3 The main elements of useML

This reduced UML class diagram indicates that a use model contains an unlimited number of UOs. Each UO can also contain an unlimited number of other UOs as well as eUOs.

Normally, the UOs and eUOs in contained in a UO are not indexed. In this case, it is a collection of logically related UOs and eUOs. But if a UO contains only indexed eUOs, then it is a task and the sequence of the eUOs indicates a method of achieving the task goal. In this sequence, a eUO can be specified as optional, otherwise, it is mandatory.

An elementary use object can be of type *select*, *edit*, *manipulate*, *command* or *inform* (see Fig. 3). These correspond to the actions of the machine user.

- *<select>*
  By this action, the user can select one or many values from a set of values that already exist in the machine 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>*
  This involves input of one absolute data value into the machine system. Although the machine can suggest a default value, it will be overwritten by user input. The machine does not know the data value (other than the default one, if available) before user input. It means that the input comes from outside the system. An example of this interaction type is "enter the user name".
- *<manipulate>*
  This interaction is basically like edit. The difference is that the system provides a way of manipulating input relative to an existing value. This means that the data already exists in the system. For example the user can increment the speed from 15m/s to 17m/s by using a Toggle-Wheel-Switch with 2 as an incrimination factor.
- *<command>*
  This is an action where the user directly triggers a machine function; the user explicitly commits a command to a machine. This results into direct execution of machine functions. For example, starting the machine by "pressing" the start button.
- *<inform>*
  Actions of the type inform involve the user querying the machine for some information. For example the user would like to know the status of the machine. It is a unidirectional action, meaning that no directly related action from the user is expected after viewing the information.

The few useML elements are easy to handle and to use since they have a usage based notation. With useML, you just need to describe the tasks as they are

performed. Consider Fig. 4 as an abstract example of some user tasks. The figure indicates which user group is allowed to perform which action, from which machine and at which location. Further, machine functions that are invoked be the actions are indicated.

| Task/Interaction | | User Group | Machine | Location | Function |
|---|---|---|---|---|---|
| **View operating statistics** | | | | | |
| | select time interval | Supervisor | all | all | T |
| | get operating hours | Supervisor | all | all | hOp |
| | get non production time | Supervisor | all | all | stopTime |
| | get total output | Supervisor | all | all | Out |
| | get output per day | Supervisor | all | all | dOut |
| | get average output per hour | Supervisor | all | all | avgOut |
| | get number of production interruptions | Expert | all | all | iCount |
| | reset statistics | Expert | all | local | sReset |
| | | | | | |
| **View plant statistics** | | | | | |
| | select time interval | Supervisor | all | all | T |
| | get plant operation time | Supervisor | all | all | plantOp |
| | get plant non opertation time | Supervisor | all | all | stopOp |
| | get expected production | Supervisor | all | all | Pex |
| | get plant eficiency | Supervisor | all | all | Peff |
| | | | | | |
| select machine | | all | all | all | M |
| change revolution speed | | Supervisor | all | local | R |
| change production rate | | Operator | all | local | Pr |
| get engine temperature | | all | all | all | eTemp |
| get engine power | | all | all | all | eP |
| get medium temperature | | all | all | all | mTemp |
| start sample production | | all | all | local | sP |

Fig. 4 An example of user tasks and actions.

Two main tasks can be identified; *view operation statistics* and *view plant statistics*. Taking the task "View operating statistics" as an example, we see that it has the following actions:

1. select time interval,
2. get operating hours,
3. get non production time,
4. get total output,
5. get output per day,
6. get average output per hour,
7. get number of production interruptions and
8. reset statistics.

While the first six can be performed by the supervisor, at all machines and from any location, the last two are performed by the expert. The last action must be performed direct at the machine.

During the action *select time interval* the user can select only one time interval for the statistics he needs to view. Possible values are; *today*, *last day*, *last week* and *last month*. As expected, this selection is modeled by the elementary use object *<select>* (see Fig. 5). The name of the elementary use object is derived from the interaction without the word *select*. Hence the elementary use object has the name *Time interval*. The attribute *multiple_selection* that is associated with this elementary use object has the value *false* indicating that only one selection is allowed. The default value of this selection is *today* as indicated by its attribute *selected* which have the value *true*.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="useML_Viewer.xsl"?>
<usemodel author="Reuther (ZMMI - TU Kaiserslautern)"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="useML_schema.xsd" id="1">
    <!-- Filter options for prototypes -->
    <filter prototyp="useML_PDA" screensize="480x585"></filter>
    <!-- More filter options can be added -->
    <name>Useware</name>
    <!-- other UOs and eUOs have been deleted -->
        <useobject>
            <name>View operating statistics</name>
            <mapping>
                <machine typ="all"/>
                <user_group typ="supervisor"/>
                <location typ="all"/>
            </mapping>
            <select multiple_selection="false">
                <name>Time interval</name>
                <mapping>
                    <machine typ="all"/>
                    <user_group typ="supervisor"/>
                    <location typ="all"/> <function ID="T"/>
                </mapping>
                <option selected="true"><value>today</value></option>
                <option> <value>last day</value> </option>
                <option><value>last week</value></option>
                <option><value>last month</value></option>
            </select>
            <execute confirmation_required="yes">
                <name>Reset statistics</name>
                <mapping>
                    <machine typ="all"/>
                    <user_group typ="expert"/>
                    <location typ="local"/>
                </mapping>
            </execute>
        </useobject>
</usemodel>
```

Fig. 5 A use model showing part of the task "View operating statistics"

The use model is platform independent. Since it is XML-based, it can be easily treated with style sheets to provide prototypes for validating the structure. Fig. 6 shows an example of two generated prototypes for a Web client and a PDA. The dynamic can be simply implemented with Java script.

Fig. 6 Useware prototypes for Web client and PDA.

Such prototypes do not only help the evaluation process, but also accelerate the decision making.

### *3.2. The abstract user interface model*

After defining user tasks in a platform independent way, designing the user interface begins. While the

use model describes the structure of user tasks and their relationships, the goal now is to identify abstract interface objects (aUIO) needed to perform the tasks and map them to their corresponding use objects from the use model. aUIOs are also known as abstract interaction objects (AIO) in other papers, for example (Szekely, 1996). Naming them user interface objects has an emphasis on the *user interface* and not on the *interaction*, since they are not necessarily interactive.

The abstract user interface model is a central model for all concrete user interfaces running on different platforms. Besides defining aUIOs, it also saves the purpose of defining other aspects that should be common at all interfaces. For the example, the grouping of layout of objects, their behavior and the navigation structure. Having these aspects defined at one place helps to ease maintenance and supports consistence.

The elements of the abstract user interface can be seen in a simplified UML class diagram of Fig. 7.

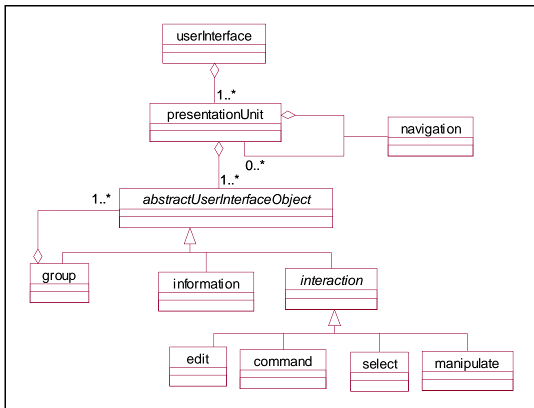


Fig. 7 Elements of the abstract user interface

The transformation of the use model into an abstract user interface model follows the following guidelines:

1. starting from the root element, each use object is mapped onto a presentation unit.
2. The connection between the use objects is stored in the navigation element.
3. the elementary use objects are mapped onto abstract user interface objects.
4. should there be need to group some eUOs, then the *group* element is used.

After the transformation, a general layout for the presentation units can be defined. This might required considering hardware constraints like display size (remember the overlapping between the structuring and design phases). By applying special ergonomic design rules for machine user interfaces, the operation panel is divided into different areas, which will contain logically identical widgets. For example, there may be an area for navigation, for function keys and for data display (VDI/VDE, 2000). The data display area differs from the other two while it contains dynamic content. It is a main area where the user can view, enter or change data. The Display area can further be partitioned into message and status areas and an area for data input and output (see Fig. 8).

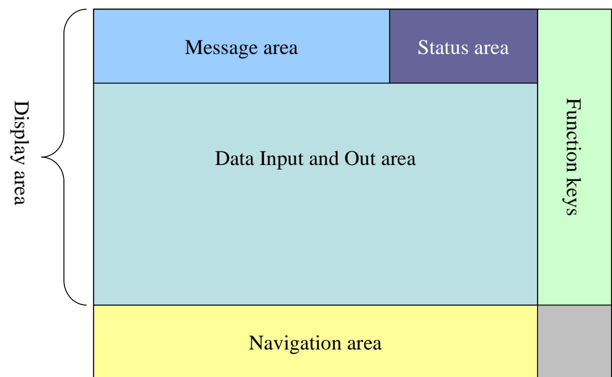


Fig. 8 An example of layout for a machine user interface.

Diagrammatic prototypes like wire-frame mockups or abstract layout diagrams can be used in this phase (Constantine *et al.*, 2003). Design can now proceed with the definition of the concrete user interface.

### *2.4 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. Concrete interface 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. 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.
Therefore, depending on the platform abstract interface objects are mapped onto specific interface objects according to the common design guidelines, for example, the type of object (selection, information, command, edit, manipulate), size of object and number of elements. An approach of implementing this mapping is shown in the TRIDENT Project (Bondart *et al.*, 1994).

## 4. RELATED WORK

The model-based approach has found interest of many researchers. This has led to the rapid development of many solutions for user interface description languages.

The *User Interface Markup Language* (UIML) is perhaps the most known XML-based UIDL. It provides platform independent elements for defining the user interface. The main elements are the *<interface>* that contains interface specific elements like *<structure>*, *<content>*, *<style>*, *<behavior>* and the *<peers>* (Abrams *et. al.*, 1999). The *<peers>*

element enables rendering the interface to a specific platform with platform specific toolkit. Though UMIL elements are platform independent, one needs platform knowledge in order to be able to define an interface for a specific platform. For example the developer must know if the structure he defines fits to the target platform. Therefore a UIML document is platform dependent. Also UIML does not support abstraction of tasks and description of common interface issues. These must be repeatedly defined for each platform, hence it is difficult to maintain and to keep consistence between descriptions.

The *eXtensible Interface Markup Language* (XIML) provides a better solution. It defines necessary requirements for a universal user interface description language as well as its structure (Puerta and Eisenstein, 2004). It proposes a solution for standardizing the representation and manipulation of *interaction data* – the data that defines and relates all the relevant elements of a user interface. In so doing, XIML is rather a framework than a description language. The basic interface components suggested by XIML have some resemblances with the models of the MB-UID (Szekely, 1996); the *task* component corresponds to the task model, the *domain* component corresponds to the domain model, the *user* component to the user model, the *presentation* component to the presentation model and the *dialog* component to the dialog model. The XIML Framework extends this by explicitly modeling relations between elements and their attributes by using the *relations* and *attributes* representational units. Though it is a good idea to standardize the representation and manipulation of interaction data, a concrete XIML-based implantation of the framework could not be found (at least it was not included in the paper). It is therefore not clear whether XIML as a language really exists.

Also, a number of model-based development processes can be found. The most recent that was found is TERESA (Mori *et. al.*, 2004). It provides both a development process and a description language. Starting from a nomadic task model, several models must be defined. For each platform three different models, namely; a *system task model*, an *abstract user interface model* and a *concrete user interface model* must be defined. These many models are difficult to manage, especially when developing for multiple devices.

## 5. CONCLUSION

This paper has presented a systematic Useware development process accompanied with a model-based approach as a solution for user interface development problems today. The process consists of overlapping phases with continuous evaluation. This can ensure maximum end user participation in the development process.

Also the paper has pointed outs that a description language should be easy to use and capture domain specific aspects, if it has to find wide applicability within that domain. The paper has introduced useML as an example in the domain of production automation. It consists of five task based objects; *select*, *edit*, *manipulate*, *command* or *inform*. The useML based development makes it possible to specify the hardware and software of the interaction-system in the late development stages, leaving the early development stages platform independent. In this way, the same *use model* can be tailored to any target platform. A tool to support the development process and useML is being developed.

## REFERENCES


Abrams, M. *et. al.* (1999). UIML: An appliance-independent XML user interface language. In: *Proceedings of 8th International World-Wide Web Conference WWW'8.* Mendelzon, Toronto.

Bondart, F. *et al.* (1994). A Model-Based Approach to Presentation: A Cpntinuum from Task Analysis to Prototype. In: *Proceedings Design, Specification and Verification of Interactive Systems*, pp.77-94, Springer Verlag.

Constantine, L. *et al.* (2003). From Abstraction to Realization: Canonical Abstract Prototypes for User Interface Design. *Working Paper Vers. 2.0,* http://www.foruse.com/articles/canonical.pdf.

Mori *et. al.* (2004). Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. *IEE Transactions on Software Engineering*, *Vol. 30*.

Mukasa, K. and A. Reuther. (2004). The Useware Markup Language (useML) – Development of user-centered Interfaces using XML. In: *9th IFAC/IFIPS/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine Systems*, Atlanta.

Puerta, A., and Eisenstein, J. (2004). XMIL: A Universal Language for User Interfaces, www.ximl.org/documents/XimlWhitePaper.pdf.

Reuther, A. (2003). *useML – Systematic Useware-engineering with XML.* Kaiserslautern University of Technology, Kaiserslautern.

Szekely, P. (1996). Retrospective and Challenges for Model-Based Interface Development. In: *Computer-Aided Design of User Interfaces*, pp. xxi-xliv.

VDI/VDE3850 Norm (2000). *Nutzergerechte Gestaltung von Bediensystemen für Maschinen (user-friendly design of useware for machines).*