Tool Support for model-based Generation of Advanced User-Interfaces

Andreas Wolff, Peter Forbrig
University of Rostock
Institute of Computer Science
Albert Einstein Str. 21,
18059 Rostock, Germany
[rusty|pforbrig]@informatik.uni-rostock.de

Daniel Reichart
University of Rostock
Institute of Computer Science
Albert Einstein Str. 21,
18059 Rostock, Germany
dr007@informatik.uni-rostock.de

ABSTRACT
A lot of research and work has been done in the past, to develop XML based user-interface definition languages. Also languages to describe graphics and animations were created.

In an attempt to combine two of those languages with model-based user-interface generation, we demonstrate an idea of how a specific toolset, originally developed to support UI-Pattern based development, can be used to create advanced user interfaces based on models.

Author Keywords
Model-Based Design, Task Models, Patterns, XUL, XIML, SVG.

ACM Classification Keywords
HCI

INTRODUCTION
Model-based development of software systems has become popular in particular along with the enhancing capabilities of mobile devices. In this domain it is especially necessary to design user interfaces in an abstract way, because there is a diversity of different platforms with specific features. These varying platforms have to be supported in an economic way by new interactive applications.

We consider model based software development as a sequence of transformations of models that is not performed in a fully automated way, but supported by humans using interactive tools. Figure 1 provides a graphical representation of that approach.

It is our opinion that software engineers and user interface designers; have to base their work on the same models. Our work is especially focused on methods and tools supporting transformations by patterns.

In the past a number of tools were developed to support different aspects of the MDA approach mentioned above. In this paper we focus on those tools that assist designers in the user interface development process.

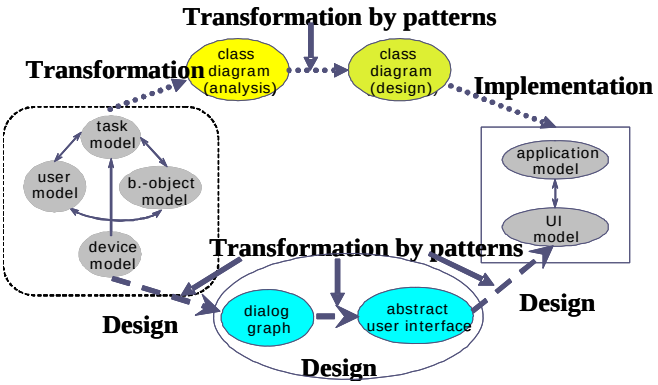

Figure 1 - General view on a transformational model-based development process.

A proposal is presented, how two of our tools could be used in combination to generate and design advanced user-interfaces that are based on and derived from models.

This paper is structured in such a way that after discussing some related work our envisioned development process is presented, which includes a short introduction to the supporting tools.

Afterwards a small sample user-interface is developed that demonstrates the capabilities to create an advanced user-interface based on models. This paper does not document an already achieved state of work, but is meant to discuss the idea. An overlook on remaining future work is given at the end.

RELATED WORK
Our work is generally related to the “mapping problem” that was first mentioned by Puerta and Eisenstein. They stated that the mapping problem is the key problem to make model-based development acceptable for programmers. The mappings mentioned include mappings from abstract to concrete models and between models of the same level. Mappings from concrete to abstract models are considered by Clerckx, Luyten and Coninx in [1]. This level of mappings is of minor relevance for the purpose of this

paper. Mapping from concrete models to abstract ones is more important.

Limbourg and Vanderdonckt [5] address the "mapping problem" by supporting transformation of abstract models to more concrete ones by graph grammars. The user interface specification is based on UsiXML [8].

UsiXML – USer Interface eXtensible Markup Language – is a XML language for describing the UI for multiple contexts of use. It can be used for Graphical-, Character-, Auditory-, and Multi-Modal User Interfaces. A UsiXML UI-description is independent from an underlying computing platform. Currently it seems to be that UsiXML could be a living standard to express models.

UsiXML possibly can play the role, which XIML [10] originally wanted to gain. The initiative for XIML started in 1999 and was focused on device-independence primarily of mobile devices. XIML is model-based but it needs a specific tool to create a specific type of user interface. Our tool DiaTask was developed to make use of XIML. Within DiaTask task models, user models, and object models with our metaphor of artefacts and tools are represented as XIML specifications. However, there seems to be no further support for XIML. Still there is a lack of tool support, for example for designing a concrete user interface. That was the reason for our group to look for user interface specifications, which are already supported by tools. We found XUL as a candidate for that.

XUL was presented in 1999 by the Mozilla project to specify Graphical User Interfaces of the Mozilla-browser in platform-independent matter. XUL allows the specification of interactive objects like buttons, labels, and text fields.

SVG [12] – Scalable Vector Graphics – was standardized in 2001 by W3C, as a XML based language for the description of animated and static 2-dimensional vector graphics. For use in mobile devices there are two separate standards *Tiny* and *Basic* which contain only a subset of SVG. SVG can be embedded into XUL user-interface definitions and by making use of defined scripting languages it is possible to create advanced user-interfaces.

To design concrete user-interfaces a GUI editor for XUL – called XUL-E – was developed [9], it is based on an already existing Eclipse plugin. XUL-E was built in such a way that co-operation with task models and generated user interfaces became possible.

In the following XUL-E and DiaTask will be used to generate a XUL/SVG user-interface from a task-model.

DEVELOPMENT PROCESS OF A USER INTERFACE

According to Figure 1, the development of an application starts with (1) creating different models. Based on them a (2) dialog-graph is created, which is used to find an (3) abstract user-interface as basis for (4) concrete user-interfaces. Transformations between those steps should be done by using patterns.

We try to establish an evolutionary approach that covers all four phases. This consists of a mechanism to reflect changes to one model in other affected models. Such mechanism enables us to let each model evolve step-by-step but keeping consistency between the models. Indeed, it becomes possible to integrate a new task into an already existing concrete user interface, without going through all the previously mentioned steps again. We do not believe that this can be done in a fully automated manner, but will require user intervention.

For now we will concentrate on task-models as initial models of an application. Typically task-models get designed in CTTE-notation [2]. For an example see Figure 2.

As second step a pattern based transformation should result in a dialog-graph (see e.g. [4]). Currently there is no satisfying way of doing this. However, one can use our tool "DiaTask" to generate dialog-graphs manually.

By using DiaTask, a user at first has to decide, how many views are desired for an application, and whether each of them is a modal, single or multiple one. Next step is to assign relevant tasks to views. The task model of the application determines the set of tasks, which can be distributed on views. Thereafter a designer has to model transitions between tasks and views. DiaTask does support necessary operations to do this.

Given a dialog graph DiaTask can generate an initial abstract user-interface prototype in a WIMP style that mainly reflects the navigation structure of the user interface. Windows are used to represent views and elements of the views are mapped to buttons. Other task-element mappings can be achieved by applying a different presentation model. This generated abstract user-interface is currently stored in XUL format. At this point it is already possible to animate the AUI, which can be useful for testing purposes.

Following the generation of the abstract user interface, in a next step a concrete user interface (CUI) is to be designed.

We support this step with our XUL editing tool (XUL-E) [9]. Beside its graphical editing features its main purpose is to support our evolutionary approach. For that some information exchange between editor and DiaTask is necessary. This is handled by a slightly enhanced version of the XUL language, which is called XULM. These enhancements include task-control data, administration data and elements to support UI-patterns.

XUL-E uses DiaTask's generated AUI as starting point for layout refinements. The basic idea of an integrated editing process, as presented here, is to edit by replacements. To design the user interface for a certain task, one replaces its current visualization by another one.

Replacements are done interactively by "drag & drop" and can be either a simple graphical element, e.g. a checkbox,

or a pre-designed component. This controlled replacement process enables XUL-E to maintain any task-related attribution of an element and accordingly keep connections to task and dialog model.

For example, in the context of a mail application, a minor task as “Choose always send receipt-notification” could be replaced by a checkbox; while more complex task, such as “Save attachments”, might be replaced by a component consisting of a list-box and some control buttons. It depends on the abstraction level of each single task and the availability of pre-designed components.

Replacing a single graphical element, such as a button of the initial AUI, by a more complex component raises the problem of where to attach task-related information. XULM offers fine-grained control on this matter. It is possible to define for each element inside a pre-designed component whether it should have task-control-data applied or not.

For logical and organizational reasons XULM offers to group components into packages. Those packages again can contain packages, creating a hierarchy in this way. The main idea is to group different visualizations for the same kind of task(s) into one (sub-) package. Those visualizations should cover different contexts-of-use. An application AUI will carry references to such packages, so applying a user interface of an application to a different context-of-use is ideally reduced to referencing a specific component of the same package.

After finishing the replacement process for each view, it is possible to animate the resulting concrete user-interface within DiaTask or another XUL interpreter. As already mentioned, XUL-E is embedded within our evolutionary process. For details see [11].

SAMPLE APPLICATION

In the following the outlined approach is used to generate a very small sample application whose concrete user-interface will make use of SVG and XUL.

Context of the application is a garage. It shall enable its users, which are mechanists, to select a broken engine part on a screen, optionally further describe the failure and then order fitting spare parts from a distributor. An initial task-model, in CTTE-notation, is presented in Figure 2.

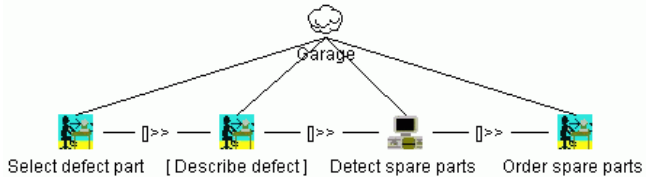

Figure 2 – CTT for sample application

The created task-model is used as input for DiaTask, as described earlier. After assigning, user affected, tasks to views the result might be a dialog-graph as in Figure 3. Note, that no view got task “Detect spare parts” assigned, as it is a task without user interaction.

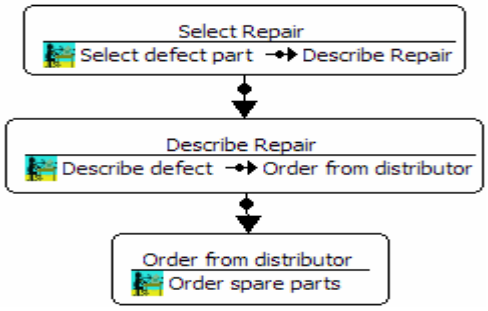

Figure 3 – Dialog-graph for sample application

The generated abstract user-interface would consist of three small XUL windows, each containing a labelled button, with the name of the assigned task. It is not displayed here, because of lack of space.

Last step in creating a concrete user-interface for this application is done by using XUL-E. We will design the view “Select Repair” to contain an SVG designed engine. SVG itself cannot be created by the editor; it needs to be provided by external editors, e.g. by “Inkscape” [13].

To integrate it into a view, XUL-E’s components mechanism is used. An engine first needs to be drawn and is afterwards stored as a component in a package to which the editor has access to. If desired, different SVG drawings for an engine can be stored into that package, e.g. one using all elements of standard SVG and another one using only elements of SVG *Tiny* to support PDA or cell phones.

SVG supports zooming and multi-level drawings. A mechanist is expected to define the broken part of the engine by zooming in and selecting it within the SVG-image. Therefore application logic, to determine which part is selected, might be needed. It would be written in a script language and should possibly be embedded within the resulting XML concrete user-interface description. This can be achieved by putting those scripts into another component and add it to the designed interface.

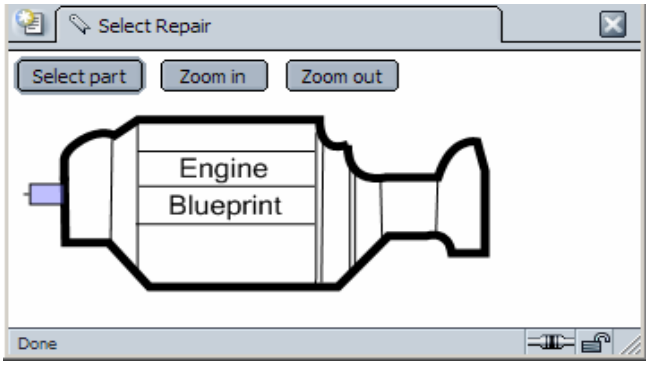

Figure 4 – “Select Repair” view as XUL/SVG combination, interpreted by Mozilla

As the editor and XULM are designed in respect of applying UI-patterns to an abstract or concrete user-interface, XULM offers to logically combine multiple components to a single one. Such a single component is required by XUL-E for its replacement process, as mentioned above. The combination and replacement process can further be controlled on XULM and editor level, to allow semi-automatic adaptation to different contexts of use. This process is omitted in this paper.

Figure 4 shows the view with a sketched engine, a select and two zoom buttons as part of the applications control logic component. SVG is not limited to such simple graphics; it is only a simplified example for demonstration purposes.

The view "Describe Repair" is designed with a textbox, e.g. for inputting key numbers, and the task "Describe defect" itself is replaced by an "OK-Cancel"-component. Inserting the functionality "Cancel" requires adapting the dialog-graph with a transition back to view "Select Repair".

View "Order from distributor" is replaced by a pre-designed component "mailform", which contains a layout of XUL elements to write an email and an "OK-Cancel"-component. Adaptation of the applications dialog-graph is therefore also required. The result of the transformations for those views is displayed in Figure 5.

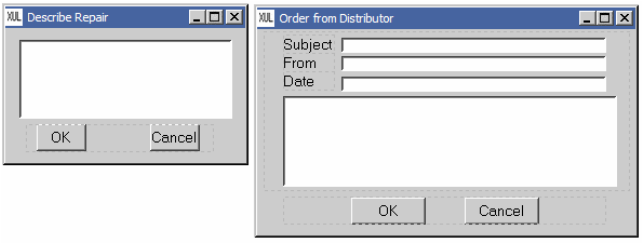

Figure 5 – Describe and Order views in XUL-E

CONCLUSION AND FURTHER WORK

We have shown that the combination of the DiaTask and XUL-E tools can support model-based development of user interfaces. Starting with a task model the designer can interactively create and design an abstract and concrete user interface for an application. As the resulting specifications for abstract and concrete UI are in XUL format, there is a chance of converting them into native code for programming languages by using specialized compilers.

Furthermore we proposed a method for integrating SVG into the overall process. Contained within components vector graphics can be included into a concrete UI just as any XUL element or component. Using features of XULM there is a fine-grained control over the resulting concrete user interface and it also is adaptable to different contexts of use.

As SVG offers animation, zooming and can be accessed via a scripting language, which itself can be embedded within XULM components, it is possible to create advanced user-interfaces. In order to demonstrate the application of our approach the development of the UI of a small sample application was presented.

With our current toolset we can support the operations, which were needed to generate the results of Figures 4 and 5. However, momentarily viewing and editing SVG graphics within XUL-E still remain manual activities.

Another limiting factor is XULM; basic functionalities such as tracking tasks over models and views are already implemented. Defining components using XULM, especially in term of UI-patterns, is still under development.

REFERENCES

1. Clerxkx, T.; Luyten K.; Conix, K.: The Mapping Problem Back and Forth: Customizing Dynamic Models while preserving Consitency, Proc. TAMODIA 2004, P. 33-42.
2. CTTE: The ConcurTaskTree Environment. http://giove.cnuce.cnr.it/ctte.html.
3. Dittmar, A., Forbrig, P., Heftberger, S., Stary, C.: Tool Support for Task Modelling – A Constructive Exploration. *Proc. EHCI-DSVIS'04*, 2004.
4. Elwert, T., Schlungbaum, E.: Dialogue Graphs – A Formal and Visual Specification Technique for Dialogue Modelling. In Siddiqi, J.I., Roast, C.R. (ed.) *Formal Aspects of the Human Computer Interface*, Springer Verlag, 1996.
5. Limbourg, Q., Vanderdonckt, J.: Addressing the Mapping Problem in User Interface Design with USIXML, *Proc TAMODIA 2004*, Prague, P. 155-164
6. López-Jaquero, V.; Montero, F. ; Molina, J.,P.; González, P.: A Seamless Development Process of Adaptive User Interfaces Explicitly Based on Usability Properties, *Proc. EHCI-DSVIS'04*, p. 372-389, 2004.
7. Paterno, F.; Mancini, C.; Meniconi, S: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models, *Proc. Interact 97*, Sydney, Chapman & Hall, p362-369, 1997.
8. UsiXML: http://www.usixml.org/
9. Wolff, Andreas: Ein Konzept zur Integration von Aufgabenmodellen in das GUI-Design, Master Thesis, University of Rostock, 2004.
10. XIML: http://www.ximl.org
11. Wolff, A.; Forbrig, P.; Dittmar, A.; Reichart, D.: Linking GUI Elements to Tasks – Supporting an Evolutionary Design Process, accepted for *TAMODIA* 2005, Gdansk
12. SVG: http://www.w3.org/Graphics/SVG/
13. Inkscape: http://www.inkscape.org