# Model-based Reengineering of User Interfaces

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

**ABSTRACT**
This position paper shortly describes methods and tools under development to support a model-based reengineering process of user interfaces of legacy applications. This reengineering process enables the use of HCI patterns and allows an adaption of user interfaces to different contexts of use.

**INTRODUCTION**

During the last years a number of approaches were proposed of how to design and create software and user interfaces in a model-based manner. Many of them can only be used for newly designed projects. Figure 1 gives an impression of the general idea of this model-based UI development process. Details on those tools and methodology are for example found in [10].

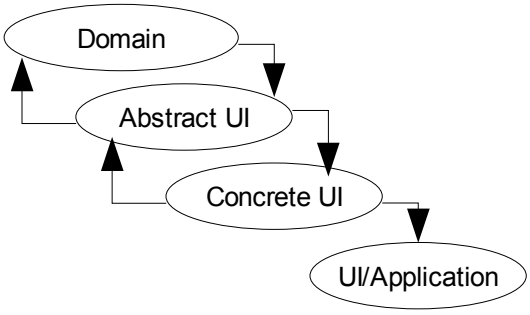


**Figure 1: Rough sketch of basic MD-UID principle**

Figure 1 sketches a generally agreed principle of model-driven user interface development (MD-UID) for singlemodal UIs. A connection from "UI/Application" back to "Concrete UI" was omitted deliberately, as we don't consider UI-reengineering to be part of the general approaches.

It is a problem to apply the model-driven development (MDD) to already existing software systems. We aim to open MDD for legacy software as well. In this paper an approach is presented of how to extract a model of an application's user interface from the application itself. We start bottom-up by extracting a model of its concrete user interface (CUI). This model can be abstracted into an abstract user interface (AUI). While abstracting, connections to business logic and object model are preserved. Object model thereby refers to the class-definitions of runtime objects. Figure 2 illustrates the general idea of our reengineering process. Dotted lines denote references, solid lines show the process flow.

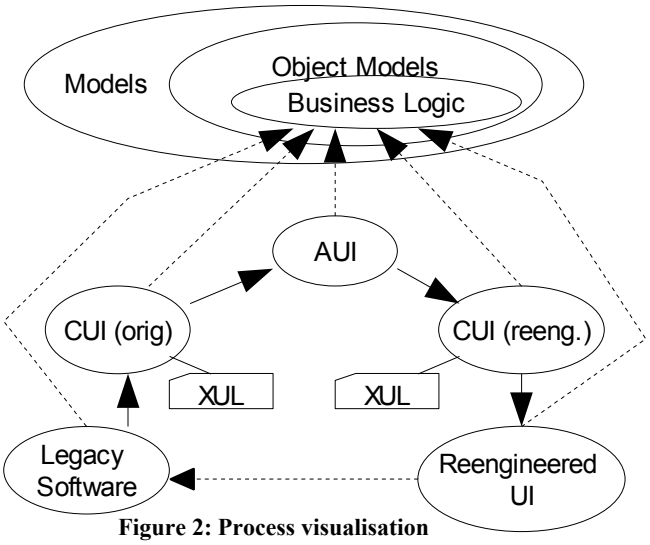


**Figure 2: Process visualisation**

By using AUI and CUI models we can transform and adapt the user interface to change appearance or concept, or to apply HCI patterns. Since we keep business-logic and object-model references, a generated user-interface from these models ought to be able to interact with the original software. In practice, we currently only are able to perform this process for very basic applications and in a restricted environment.

A lot of our tools are based on XUL and written in Java. This also applies to the reengineering tool-set. Hence this tool-set was intentionally planned to only examine and re-generate Java-Swing applications. We don't consider this as a confining restriction, but rather as a necessary simplification to actually deliver a sizeable proof-of-concept application in appreciable time.

Comparable work of extracting user interface models from existing applications and reengineering was also done by Vanderdonckt et. al. [8,9]. They focussed on web-pages and applications written in the programming language C(++).

Other related work was done by Mainetti et.al. [11]. They are focussing on the redesign of web applications, also including code analysis to derive object models and business logic.

To assist the reengineering of character based user interfaces to graphical user interfaces a model based approach was proposed by Tucker and Stirewalt [13]. It focusses on batch-command systems and utilizes an intermediate model comparable to data flow diagrams to determine required user input and application flow. It seems to have been abandoned though.

Another of our main research topics is to integrate HCI patterns into MDD. Object oriented design patterns, as introduced by Gamma et. al., are considered as valuable aid in software development. We expect comparable benefit of HCI patterns as well and therefore try to integrate those patterns into our model-based UI development process.

The main principle to achieve this is to translate HCI patterns into machine-readable attributed components when and where appropriate. We call these components "pattern instance components (PIC)" [7] as they contain an instance of their respective pattern. PICs are used to semi-automatically transform parts of abstract or concrete interfaces to follow a certain pattern.

Arnout's [1] research on object-oriented pattern components inspired our proceeding, which is e.g. described in [2].

## EXTRACTING USER INTERFACE MODELS

To extract UI models from Java applications a tool "Swing2XUL" is under development. Swing2XUL is a wrapper application. It registers listeners for any frame that is instantiated by a running application. By entering a special keyboard command it converts the contents of any currently active frame to XUL [6], see figures 3 and 4 for an example.

Our demonstration application is simple. It has two labelled input fields for entering first name and surname. On pressing "Propose User Id" a new string is calculated by concatenating the first three letters of both name parts. Afterwards this string is converted into lower-case form and presented as "User-Id" in its own text field.

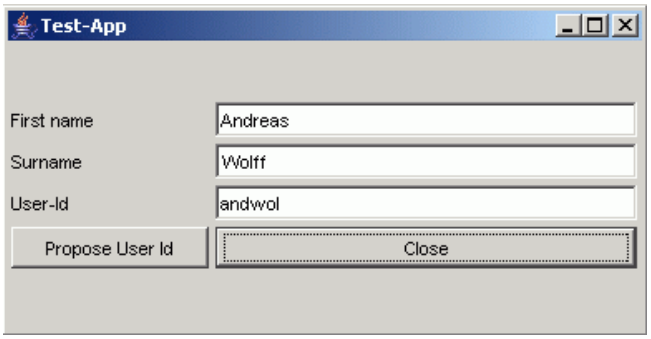


**Figure 3: Java UI of sample application**

Figure 4 depicts how SeaMonkey's Gecko engine interprets Swing2XUL's parsing result of the frame shown in figure 3. We consider this XUL result as CUI of our application. Taking it as input we are able to derive an AUI of this application in an UIML [12] related dialect. See USGP in [3] for details, we do not want to discuss that matter in this paper.

## BUSINESS LOGIC AND OBJECT MODEL

Converting a Swing-UI to a similar looking XUL-UI is not difficult. Major problem is to get an object-model and keep the linkage from UI elements to that. Those references are needed later to synthesize a running application.

In a simple first attempt we tried to make use of XMI-UML models, generated from an application's Java source code by ArgoUML [5].

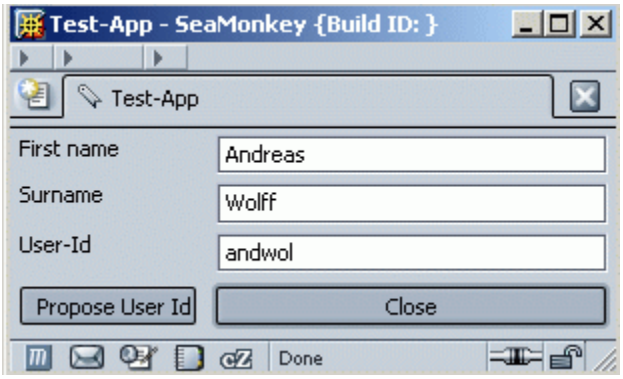


**Figure 4: XUL rendering of Swing2XUL's result**

Swing2XUL, per introspection, can identify the runtime-class of any triggered action and displayed object and by that connect it to our object-model. Such a reference basically is a pointer to an element within our XMI-UML. This information, i.e. the detected reference, is stored into our generated XUL-GUI. It is stored as an attribute, within an own namespace, in its respective XUL-element.

For the purpose of reengineering our sample application the relevant references would point 1) to an action to calculate a user id and 2) to a close action. Both are annotated within their XUL button elements.

Check boxes, radio button and menus are handled in a similar manner. More complex references are needed for tables and trees to consider their respective internal models, renderers and editors.

## REENGINEERING

At this point we have a concrete user interface and our object-model. Thus the reverse engineering of our sample application, with respect to model based development, was successful.

We now are able to rearrange the user interface of our sample application and to generate a new running application with a different UI from that.

Assuming that usability testing has shown that users want more guidance through the application, we decided to apply the "Wizard" pattern [4] to it. This is a semi-automatic process.

A human designer has to decide which elements of the UI are to put in each single wizard page and also their sequence.

Application of patterns to a concrete user interface and editing of XUL based CUI specifications is done by a specialized editor named XUL-E [2].

PICs form the basis of pattern application. As mentioned before these are components consisting of transformation rules. Those rules allow the modification of concrete user interfaces in XUL based on an abstract model description, in this case "Wizard's" details.

An application of PIC "Wizard" generates a number of views, or pages, that are interconnected by "Next", "Prev" and "Finish" controls.

Sequence and content of pages are determined by multiple area selections and their sequence. This is an interactive process. At first one draws a selection rectangle around the planned content for page one, afterwards one draws another selection rectangle around the future content of page two, etc.

Figure 5 illustrates what decisions were made for our example. Section "I" denotes all UI elements selected for the first wizard page, "II" elements for the second page and "III" elements for the third page.

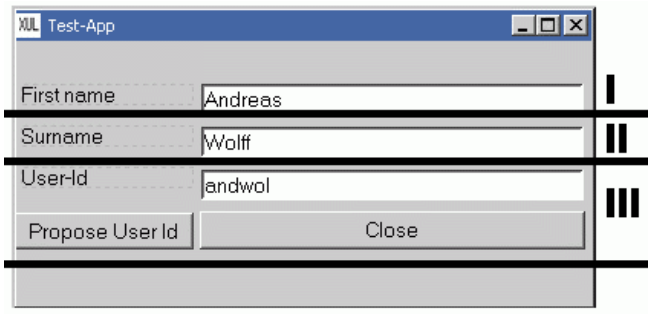


**Figure 5: Designer decision for wizard pages**

After conducting every necessary selection in proper sequence and applying the PIC "Wizard" XUL-E creates three additional windows, or wizard pages, as it is shown in figure 6.

This resulting CUI can be edited as in any other UI editor. The third view, counted top-down, does not meet our requirements and will be edited.

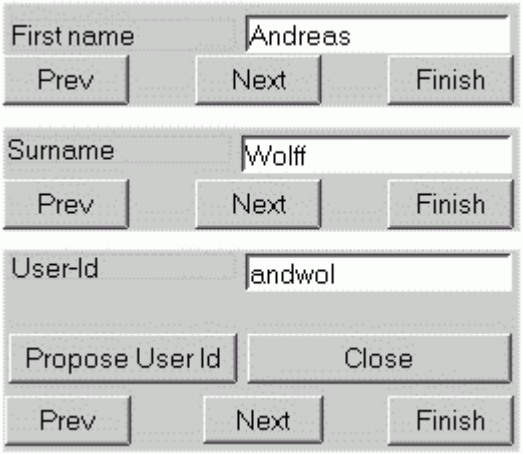


**Figure 6: Result of PIC transformation**

It was modified manually by removing "Next" and "Finish" buttons and moving the buttons from the original application into their positions. This results in a user interface as shown in figure 7 for that wizard page.


**Figure 7: Manually redesigned UI of 3rd page**

## GENERATING THE REENGINEERED APPLICATION

Our new user interface is still only displayable by XUL interpreters and it is not possible for them to interact with the original application.

We now need to generate a Java GUI that resembles our designed CUI and re-establishes connections to business logic and object-model.

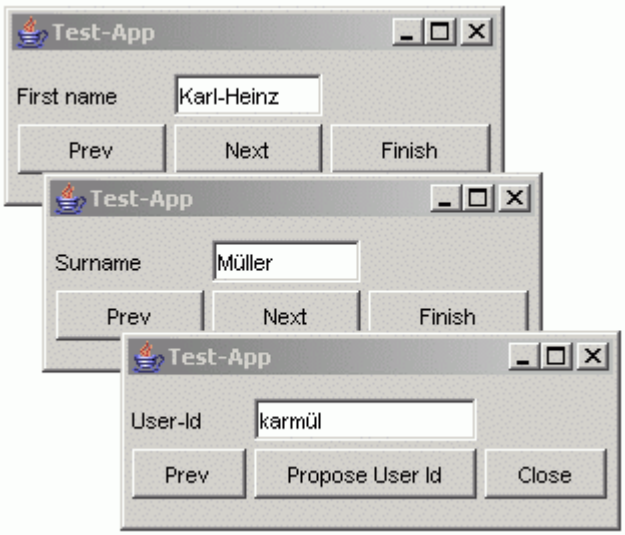


**Figure 8: New User Interface of our sample application**

There are already existing XUL interpreters for Java. For runtime interpretation as well as code generators. Possibly we might use them in a later evolution of our tools. But for

now we ourself created a small XUL to Java code generator, by using the interpreter engine of XUL-E. In this way it was easier to attach hooks for the original application's source code.

However, our code generator merges the newly designed CUI with object-model and business logic. It compiles the resulting source code and creates a new application that behaves like the original but with a different user interface.

Figure 8 is a screenshot of our new application, a user id "karmül" for "Karl-Heinz Müller" was generated.

## CONSTRAINTS

Presently our approach is not general enough to meet requirements of real-world applications. It requires the current GUI of an application to completely obey the Model-View-Controller concept of Swing, which most Java/Swing applications do not. If any data object or any part of business logic has associations to UI elements our approach eventually fails.

Also issues of technical nature do exist. For example threading and introspection done by the application itself.

Another noteworthy constraint is to have an applications source code, respectively its UI relevant part. In most cases this limitation can be overcome by using Java decompilers.

## SUMMARY & FURTHER WORK

We briefly reported on our ongoing work in the field of model-based reengineering of already existing, non-model-based developed, applications.

The basic idea is to 1) derive a CUI from a running application, 2) derive its object-model and business-logic via its source code and 3) keep connections of CUI elements to those models of step 2. Using those models we can redesign the user interface and regenerate a functional application thereof. A short sample of this process, utilizing a pattern instance component, was presented.

Further work needs to be done to map more Swing components to XUL, most importantly trees and tables. And in doing so being able to derive more complex CUIs. Also some of the current constraints need to be resolved.

## REFERENCES


1. Arnout, Karine: From Pattern to Components, PhD dissertation, Swiss Institute of Technology, Zurich 2004
2. Wolff, A.; Forbrig, P.; Dittmar, A.; Reichart, D.: Tool Support for an Evolutionary Design Process using Patterns, Proc. of Workshop on Multi-channel Adaptive Context-sensitive Systems 2006, Glasgow, GB, p. 71-80
3. Müller, Andreas: Spezifikation geräteunabhängiger Benutzerschnittstellen durch Markup-Konzepte, PhD dissertation, University of Rostock, 2003
4. Wizard Pattern, http://designinginterfaces.com/Wizard
5. ArgoUML, http://argouml.tigris.org
6. XUL XML User Interface Language: http://www.mozilla.org/projects/xul
7. Rathsack, Wolff, Forbrig: Using HCI-Patterns with Model-based Generation of Advanced User-Interfaces, Proc. of MDDAUI 2006, Models 2006, Genova, Italy
8. J. Vanderdonckt, L. Bouillon, N. Souchon, *Flexible Reverse Engineering of Web Pages with VAQUITA*, in Proceedings of IEEE 8th Working Conference on Reverse Engineering WCRE'2001 (Stuttgart, 2-5 october 2001, IEEE Press, Los Alamitos, 2001, pp. 241-248.
9. L. Bouillon, J. Vanderdonckt, J. Eisenstein, *Model-Based Approaches to Reengineering Web Pages*, in Proceedings of 1st International Workshop on Task Models and Diagrams for user interface design TAMODIA'2002, INFOREC Printing House, Bucharest, 2002, pp. 86-95.
10. Pleuß, A., Van den Bergh, J., Hußmann, H., Sauer, S and Boedcher, A.(Eds.). *MDDAUI '06, Proc. of the MoDELS'06 Workshop on Model Driven Development of Advanced User Interfaces*, CEUR Workshop Proc. 214. CEUR-WS.org, 2007.
11. Mainetti, Paiano, Pandurini: User-Centered reverse engineering: Genesis-D project, in proceedings of Web Maintenance and Reengineering 2006, CEUR Workshop Proc. 193
12. UIML, User Interface Markup Language http://www.uiml.org
13. Tucker, K. and Stirewalt, R.: Model based userinterface reengineering, in Proc. 6th WCRE, 1999. http://citeseer.ist.psu.edu/tucker99model.html