RAPIDLY MODIFYING MULTIPLE USER INTERFACES OF ONE APPLICATION
Leveraging Multi-level Dialogue Refinement

Alexander Behring, Andreas Petter and Max Mühlhäuser
FG Telecooperation, TU Darmstadt, 64283 Darmstadt, Germany

Keywords: UI models, Refinement, Engineering user interfaces, User interface description languages.

Abstract: An increasing demand for supporting users in diverse contexts of use, e.g., depending on interaction device and primary task, results in new challenges for User Interface (UI) development. Two key challenges are: how to create these multiple UIs for one application (creation challenge), and how to consistently modify them (modification challenge). The creation challenge has been addressed in various works utilizing automatic UI generation. We present our approach (Dialogue Refinement) and its tool-support to address the modification challenge by allowing one modification to be applied to multiple UIs at once.

1 INTRODUCTION

Multiple causes lead to an increase in the number of User Interfaces (UIs) per application. Technological advances allow us to interact with computers in situations that could not be supported by information technology before. We can observe that more and more commercial services are already offered not only for the traditional desktop computer, but for mobile devices like the iPhone, Java-enabled cell phones or alike. Examples beyond simple train, airplane or bus schedule services are electronic boarding-passes, and the possibility to buy electronic public transportation tickets for and with mobile devices. Regarding these UIs, key challenges are how to create UIs for different contexts of use (*creation challenge*) and how to apply modifications to them, once the UIs are created (*modification challenge*).

Our approach to address the modification challenge is based on the key idea to allow one modification to change multiple UIs. In order to control the propagation of a modification, UIs are ordered in a tree. The modification is "passed" down the tree along associations between the UIs. Changing these associations allows the UI engineer to adapt the propagation of modifications to suite the problem at hand.

Our main contributions in this paper are the refinement concept (sect. 2) and its editor-support (sect. 3). A case study and preliminary user study are presented subsequently (sect. 4). Finally, we discuss related approaches (sect. 5).

2 DIALOGUE REFINEMENT

After introducing objectives for addressing the modification challenge, our approach is introduced. it is supported by a metamodel and a toolset, which are presented afterwards.

2.1 Objectives

Objective 1: *The UI engineer must be kept in the loop.*

Automatic approaches hold great potential to reduce the effort to provide multiple UIs for one application. But their interfaces are prone to usability problems and lack aesthetic quality (Myers et al., 2000; Meskens et al., 2008) when compared to manually designed UIs.

Objective 2: *The artifact edited by the UI engineer must resemble as much as possible the resulting UI.*

Working on abstract artifacts isolates the UI engineer from the concrete interface. This was a reason why User Interface Management Systems (UIMS) did not catch on, as Myers et al. discussed in (Myers et al., 2000). They discovered that expressing graphical concepts by graphical means was an important success factor for interface builders. It lowered the threshold of use and the modifications were not prone to unpredictability. The term WYSIWYG – what you

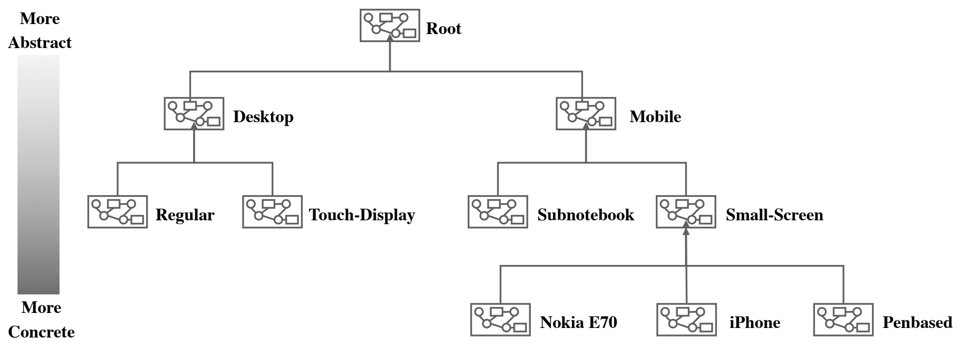


Figure 1: An exemplary *Refinement Tree* of UIs refined for different contexts of use. The topmost UI is called *Root UI*.

see is what you get – makes this even more clear: there is no difference between the edited artifact and the result.

Objective 3: *The UI engineer must be able to provide her information at the level of abstraction suitable for the problem at hand.*

UI changes can be situated at different levels of abstraction, e.g., just for the concrete iPhone UI or more abstract for all small-screen UIs. Depending on domain and application, the number of abstraction levels suitable may vary.

2.2 Concept

To create a UI for a new context of use, the UI engineer refines an already existing one and adds, removes or modifies elements of it. Applying this step multiple times results in a *refinement tree*, as shown in figure 1. The tree structure is captured through *refinement associations*, created during the refinement. These associations connect the refined UI with the more abstract[1] UI.

Our key concept to address the modification challenge is to allow one modification to be applied to multiple UIs. When applying a modification, the refinement associations are used to propagate the modification through the refinement tree, thus applying it to multiple UIs at the same time. Through changing the refinement associations, the UI engineer can influence, to which UIs a modification is propagated. Consequently, a value is either *refined* (i.e. the value is inherited from the more abstract UI), or it is *set locally*.

We implemented the concept as a metamodel in EMF (Eclipse Modeling Framework[2]). In this metamodel, UIs are made up of *InteractionObjects*. The UI engineer arranges them in *UIBoxes* to determine the look and feel of a UI. A UIBox contains the UI for a specific context of use and is a node in the refinement tree. Refinement associations can connect UIBoxes or InteractionObjects. The metamodel is agnostic of the concrete UI toolkit used, they are integrated through libraries.

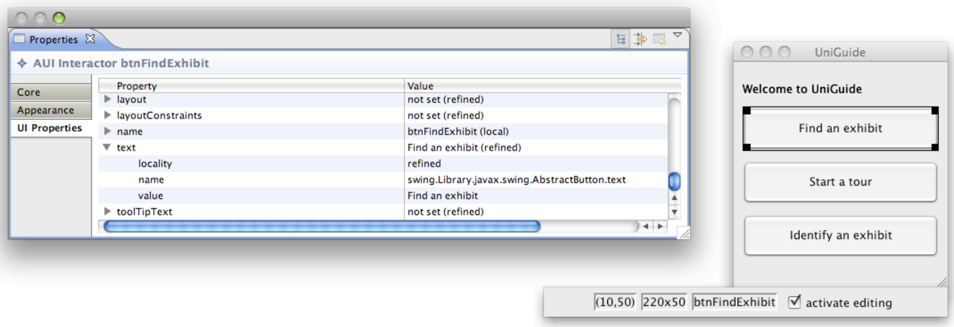
3 WYSIWYG TOOL SUPPORT

Renderers (*model interpreters*), in our approach, work directly on the UI model without intermediate code artifacts. Thus, there are no synchronization problems, which allows for an easy extension of renderers with WYSIWYG editing capability (cf. figure 2). Furthermore, editors must allow the UI engineer to modify and identify the current refinement state of a value (refined or set locally). The UI engineer must be able to explore the refinement associations for an element: does it have refining versions or an abstract version?

We implemented renderers for HTML and Swing in Java integrated into our UI research platform Mapache. The Swing renderer was extended with editing capabilities and connected to Eclipse, as shown in figure 2. The Eclipse property view allows to identify and modify the refinement state of element properties. Also in Eclipse, a *Refinement View* showing the refinement tree (cf. figure 1) was implemented. Besides showing the UI hierarchy, it can be used to explore which related elements (regarding refinement) exist for the currently selected element.

[1]We use the word *abstract* in the context of refinement rather than *generalization*.

[2]http://www.eclipse.org/emf



AUI Interactor btnFindExhibit

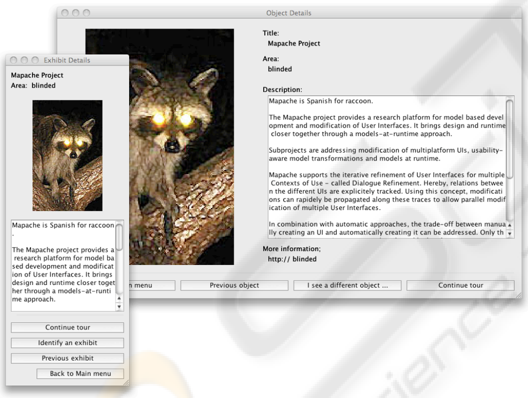
Property	Value
layout	not set (refined)
layoutConstraints	not set (refined)
name	btnFindExhibit (local)
text	Find an exhibit (refined)
locality	refined
name	swing.Library.javax.swing.AbstractButton.text
value	Find an exhibit
toolTipText	not set (refined)

(10,50) 220x50 btnFindExhibit ✓ activate editing

Figure 2: The interactive model interpreter in edit-mode with its eclipse properties page. The property page is synchronized with the interpreter and shows the currently selected item (Button "Find an exhibit"). The value of the text-property is refined from a super-version, as depicted in the properties view.

4 CASE STUDY

We created a larger case study application. The use case was a university guide (*UniGuide*), allowing people to get information on objects in our computer science building. It was targeted at three different contexts of use: desktop, subnotebook and smartphone. Figure 3 shows the "exhibit details" window refined for smartphones and desktop.



Title:
Mapache Project
Area:
blinded
Description:
Mapache is Spanish for raccoon.

The Mapache project provides a research platform for model based devel opment and modification of User Interfaces. It brings design and runtime closer together through a models-at-runtime approach.

Subprojects are addressing modification of multiplatform UIs, usability-aware model transformations and models at runtime.

Mapache supports the iterative refinement of User Interfaces for multiple Contexts of Use - called Dialogue Refinement. Hereby, relations betwee n the different UIs are explicitely tracked. Using this concept, modificati ons can rapidely be propagated along these traces to allow parallel modif ication of multiple User Interfaces.

In combination with automatic approaches, the trade-off between manua lly creating an UI and automatically creating it can be addressed. Only th

More information;
http:// blinded

Mapache Project
Area: blinded

Mapache is Spanish for raccoon .

The Mapache project provides a research platform for model ba sed development and modificat ion of User Interfaces. It brings design and runtime closer toget her through a models-at-runti me approach.

Figure 3: UIs of our case study – the "exhibit details" window refined for smartphone and desktop.

We used our editors presented in section 3 to build the UIs for the case study. Attaching behavior can be done at any refinement level using an event-registry. Domain elements are shared among different UI versions. Their properties can automatically be synced with UI element properties ("Value Bindings").

After implementing the case study, we conducted a preliminary user study. Seven participants with a great variance in GUI building skills had to complete two UI modification tasks. Task one consisted of replacing a text in all UIs, whereas task two was to add tooltips to existing elements. After a short introduction, we asked the participants to complete both tasks with and without our tools. As a baseline, we used Netbeans, because Eclipse does not provide an industrial-grade interface builder. For both tasks together, 9 steps were necessary using our tools, 21 steps in Netbeans.

All participants, even those with no experience in GUI creation, successfully completed both tasks using our approach and had no delays in applying the concept with regard to the use of Netbeans. We can conclude that the concept is easily comprehensible and thus has a low threshold of use. The normalized error rate (total errors per total number of steps) hinted that participants were less likely to produce errors when using our approach ($4\% \pm 6\%$) versus Netbeans ($21\% \pm 20\%$). We attribute this to the repetitive nature of the task in Netbeans. Such errors could pose a great problem in larger projects with a bigger set of supported contexts of use.

5 RELATED APPROACHES

Several works apply model-to-model transformations to create or synchronize UI models. For example, Sottet et al. (Sottet et al., 2006) use ATL, and Limbourg applies a generic graph-transformation language (Limbourg, 2004). Approaches that provide generic solutions, as transformations do, can be applied to a great range of problem domains. But since they are not focused on the UI, their syntax, and more important their semantic, does not address UI specific issues. Consequently, they are very abstract for the UI engineer to use. In contrast to our approach, such generic transformation approaches thus suffer from unpredictability. Furthermore, a new (often complex) language has to be learned, raising the threshold of use. In addition, they are prone to usability problems and lack aesthetic quality.

Seminal Teresa (Mori et al., 2004) supports an abstract and a concrete UI level. The semantics of abstract and concrete levels are fixed. In contrast, using our approach, the number of abstraction level can be chosen with respect to the problem at hand. Also, the UI engineer can choose the subject of abstraction freely and is not tied to fixed abstraction semantics.

Damask (Lin and Landay, 2008) is a tool mainly targeted towards UI prototyping. The modification of existing UIs, or even UIs that are already used in running applications is not Damask's focus. However, the layer concept presented in Damask is similar to the Refinement Tree of our approach. But our work goes beyond the two layers supported in Damask in order to support a wide range of different contexts of use.

Gummy (Meskens et al., 2008) is tool for creating different UIs for different contexts of use in WYSIWYG fashion. UIs for new contexts of use can be transformed from existing ones, but after creation the connection to the source UI is lost: modifications can only be applied to a single UI. In contrast, our approach keeps these connections and thus allows to apply one modification to multiple UIs.

6 CONCLUSIONS

We presented the Dialogue Refinement approach and its tool-support to address the modification challenge. The approach allows the UI engineer to apply one modification to multiple UIs at once. A case study and preliminary user study showed its applicability, ease of use, and a lower error rate compared to an industrial-grade GUI builder.

We currently research the integration of transformation approaches and the concepts presented in this paper. Because hand-crafting user interfaces for multiple target platforms is a costly task, transformations can be used for automatic generation of UIs. The aesthetic quality and usability of the UIs can be obtained by easily allowing manual modifications to generated UIs. Furthermore, we investigate the integration of a voice-based toolkit (VoiceXML) into refinement.

ACKNOWLEDGEMENTS

This work has been supported by the BMBF SoKNOS project. We thank our colleagues for their valuable feedback and Jannik Jochem for his support in implementing the tools.

REFERENCES

Limbourg, Q. (2004). *Multi-Path Development of User Interfaces*. PhD thesis, Universit catholique de Louvain.

Lin, J. and Landay, J. A. (2008). Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. In *CHI '08*, pages 1313–1322, New York, NY, USA. ACM.

Meskens, J., Vermeulen, J., Luyten, K., and Coninx, K. (2008). Gummy for multi-platform user interface designs: shape me, multiply me, fix me, use me. In *AVI '08*, pages 233–240, New York, NY, USA. ACM.

Mori, G., Paternò, F., and Santoro, C. (2004). Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Trans. Softw. Eng.*, 30(8):507–520.

Myers, B., Hudson, S. E., and Pausch, R. (2000). Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):3–28.

Sottet, J.-S., Calvary, G., Favre, J.-M., Coutaz, J., and Demeure, A. (2006). Towards mapping and model transformation for consistency of plastic user interfaces. In *Workshop "The Many Faces of Consistency in Cross-platform Design", CHI '06*. ACM.