Using HCI-Patterns with Model-based Generation of Advanced User-Interfaces

Robert Rathsack, Andreas Wolff
University of Rostock
Institute of Computer Science
Albert Einstein Str. 21,
18059 Rostock, Germany
Andreas.Wolff@informatik.uni-rostock.de

Peter Forbrig
University of Rostock
Institute of Computer Science
Albert Einstein Str. 21,
18059 Rostock, Germany
pforbrig@informatik.uni-rostock.de

ABSTRACT

In the HCI community a number of pattern catalogues were created during the last years. Due to the nature of such patterns they are often described high-level and abstract. In this paper we present an approach to translate at least a certain kind of patterns into a machine readable form, while keeping them abstract in terms of problem independence.

Those translated HCI patterns can be used for a semi-automated MDA-procedure using device- and mapping definitions. Also an example for this development cycle, a sequence of pattern-based transformations, is presented.

INTRODUCTION

Object-oriented design patterns, as introduced by Gamma et. al., are considered as valuable aid in software development. A comparable benefit is expected of HCI patterns as well. Therefore a number of pattern-catalogues were compiled by the HCI community. Well known examples are Tidwell [4] and Van Welie [5]. Along with these catalogues pattern-languages, as for example PLML [3], evolved to describe each pattern in a standardised manner.

In this paper we outline an approach of how to make use of this assembled knowledge within model-based generation of user interfaces. We attempt to represent patterns in such a way that they can be used within an (semi-)automatical MDA process to generate a concrete user interface (CUI) of an application.

Similar work has been done for the original Gamma patterns by Arnout [6], who investigated and, where possible, created usable components of design patterns in the Eiffel programming language.

We currently investigate two separate methods of where to specify the resulting "pattern instance components (PICs)". The approach presented in this paper is to place a PIC within a catalogue itself. This is possible by enhancing the underlying pattern-language and by making use of additional mapping files. The other approach is to include some sort of programming logic within the pattern catalogue itself which would reduce or even avoid the necessity of separate mapping definitions for PICs. Details about this procedure can be found in [7].

Within the first part of this paper we present the extensions to a pattern-language that we found necessary to define PICs and explain how mapping files can be used to generate CUIs from abstract user interfaces that were furnished with HCI-pattern information.

REALISATION OF PATTERN AS COMPONENTS

A pattern is an abstract description of a best-practice for a certain problem. HCI patterns in catalogues are described in a textual manner, often with a graphical example and sometimes sample source code or other implementation hints. Such a description is insufficient for MDA purposes, because it cannot be used for automated model transformations.

A pattern description usable for such transformations has to be detailed down to the abstraction of a single user interface object. Our approach is to construct PICs top-down by segmenting a pattern into smaller PICs. The bottom-level is constituted by PICs which can directly be mapped to abstract user interface objects. To refine a pattern and thus define its PIC a basic set of operators and quantifiers was identified.

XOR operator "!" is used to mark a choice between two components available as sub-PICs.

Display_Both operator "||" marks two sub-PICs as to be displayed simultaneously.

Display_Sequence operator "|-" defines two sub-PICs as to be displayed after each other. Evaluation is from left to right.

Operator	Name	Priority
!	XOR	1
||, |-	SIMUL, SEQU	2
( )	GROUP	3

Table 1 – Operator precedence

Parenthesis may be used for grouping, squared brackets to parameterise a sub-component or to pass options for the mapping stage. The operator precedence is shown in table 1.

Furthermore quantifiers were found useful to define a repeated or optional sub-component. Table 2 depicts all available quantifiers.

Quantifier	Meaning
N/A	1
?	0 or 1
*	Any, incl. 0
+	Any, at least 1

Table 2 – Impact of quantifiers

To illustrate the usage of sub-PICs, quantifiers and operators in the following an exemplary component definition is given. A pattern “Master_Detail” as generalisation of “Two-Panel Selector” and “Cascading Lists” from Tidwell will be defined. Master_Detail is applicable (1) if a user has to navigate hierarchical data or (2) if displaying detail information of a set of objects in one place is not desired or even possible, e.g. by display size restrictions. The Master-Detail pattern specifies a solution consisting of to steps: first select the object whose details are of interest and as second step display that information separately.

Attribute	Meaning
context	Name of pattern or refinement
child_rule	Available refinements and their relation defined with operators and quantifiers
applicable	Optional, restrict application of pattern or component to mentioned elements; Notation follows the one proposed in [8]
layout	Defines arrangement of sub-components in cases where child_rule defines refinements as to be displayed simultaneously

Table 3 –PIC control attributes

```
<pattern
       context="master_detail"
       child_rule="composite[default]!lookup">
  <problem>...</problem>
  <context>...</context>
  <solution>...</solution>
  <rational>...</rational>
  <related>
   <related_pattern>find/browse</related_pattern>
   <related_pattern>print_object</related_pattern>
  </related>
</pattern>
```

Listing 1 – Master_Detail as instance component

Master_Detail’s PIC is defined in an XML dialect related to PLML [3], added attributes are explained in table 3. Context, problem, solution and rational are omitted here to save space, but actually do contain a textual description. The defined component can be displayed using either PIC “composite” or “lookup”, whereas “composite” is selected to be “Master_Detail’s” default representation. Related patterns deal with comparable problems, in this case “print_object” and sub-component “browse” of pattern “find” are declared to do so.

```
<pattern
  context="master_detail/composite"
  child_rule=
  "/find/browse||(/master_detail!/print_object)"
  layout="horizontal/ungrouped"
  applicable_on="uio" />
<pattern
  context="master_detail/lookup"
  child_rule=
"/find/browse|-(/master_detail!/print_object)" />
```

Listing 2 – Master_Detail’s sub-components

Listing 2 shows the definition of Master_Detail’s possible sub-components. The major difference between them is the display sequence of alternatives. PIC “composite” shows navigation and details simultaneously on screen, while “lookup” clears the screen after selecting the target object. Note that it is possible to declare any (sub-) component type as child; this includes parent types and the current component itself. Sections from the definition of PICs “print_object” and “find” are displayed in listing 3, they require another PIC “text” of which also only a small fraction is shown.

```
<pattern
  context="find/browse"
  child_rule="structured[default]!linear" />
<pattern
  context="find/browse/structured"
  applicable_on="input_tree" />
<pattern
  context="find/browse/linear"
  applicable_on="input_1-n,input_m-n" />
…
<pattern
  context="print_object"
  child_rule="text!image"/>
<pattern
  context="print_object/text"
  child_rule="/text/multiline" />
<pattern
  context="print_object/image" />
<pattern
  context="text/multiline"
  applicable_on="input_string,output_string" />
```

Listing 3 – PIC definitions of “text”, “print_object” and “find”

At this point “Master_Detail’s” decomposition is complete, since all referenced components can be applied to abstract user interface elements.

USING PATTERN INSTANCE COMPONENTS FOR AUI

A brief description of how we integrate PICs in our model-based interface development process follows. The idea is to add component references by inserting them within a new XML namespace into a XML based abstract user interface (AUI) description language. These references are used in combination with device-dependent mappings while generating an application's CUI (concrete user interface).

We consider model-based software development as a sequence of transformations between models. Basis for any development is a task model. It results from requirements engineering and is defined by means of CTTE [10]. To derive an AUI a dialogue model of an application is developed, whereto tasks are assigned to views and transitions between such views and tasks are defined.

The combination of task and dialogue model forms an initial AUI which we currently represent in XUL [9]. To retain task references during further transformations an AUI's XUL contains XML attributes for task control data. For details see e.g. [1]

PIC references are added to the AUI definition in a similar way. An own XML namespace was created for this purpose, such it is easily possible to exchange XUL by XIML or another XML based interface language if desired in future times. Table 4 depicts a subset of content and meaning of PIC-namespaces elements.

Attribute	Meaning
pattern	PIC to use
display_sequences	Interface elements which are opened after each other, referenced by an id; correlates with "child_rule" of table 3
display_order	Defines sequence of child elements within a grouping container. Primarily useful as hint to a CUI generator.
layout	Layout hint for simultaneously displayed elements; may be used to override "layout" of a PIC definition

Table 4 – PIC reference attributes in an AUI

To demonstrate the application of a PIC to an AUI, listing 4 shows a section of a pattern enhanced AUI definition. Task control data is omitted due to space limitations. The PIC-namespace is "hcipattern".

```
<vbox id="vbox_0"
  hcipattern:pattern="multivalue_input_form"
  hcipattern:layout="vertical/ungrouped">
  <groupbox id="groupbox_1"
    name="Create new account"
    hcipattern:layout="vertical/grouped">
    <box id="grid_2" hcipattern:pattern=
      "multivalue_input_form/input"
      hcipattern:layout="table[numColumns:2]">
      <label id="label_3" name="Name:"
        hcipattern:pattern="text/label">
```

Listing 4 – PIC enhanced AUI definition

GENERATING THE CUI

In order to make use of instance components and their refinements in an automated way and therefore supply suitable tool support, the definition of mapping rules is necessary.

Mapping rules serve as and are based on device models and as such enable us to adapt a abstract user interface for different devices and contexts-of-use.

To map AUI elements to specific elements of a CUI attributes *pattern* and *layout*, of table 4, are most relevant. A mapping definition for a certain device is straightforward. In a simple XML based UI language it consists merely of value pairs (context, target). Context denotes the (sub-)PIC and target is the destination element in the CUI. To suit for more complex UI definitions parameters can be defined and consequently passed to the CUI generator as (name,value) pairs.

Should a PIC definition contain choices (XOR operator) the mapping rules also include the decision which option is to be applied for the CUI.

If a destination language already supports a layout type by itself layout-mappings should be defined. Abstract layout "vertical/grouped" in XUL as CUI for example can be mapped to a specific element "groupbox", parameterised with orientation "vertical". While an abstract layout "horizontal/ungrouped" would be mapped to an XUL "hbox" element.

Listing 5 presents a section of mapping rules relevant for "master_detail"; result would be a XUL CUI suitable for a personal computer. A rule set for XUL on PDA is slightly different.

```
<pattern_profile name="XUL/PC excerpt"
                 device="xulpc">
  <pattern_mappings>
    <mapping context="master_detail/composite"
      layout="horizontal/ungrouped" />
    <mapping context="print_object/text"
             target="textbox">
      <feature name="multiline" value="true" />
    </mapping>
    <mapping context="print_object/image"
             target="image" />
    <mapping context="find/browse/structured"
             target="tree" />
  </pattern_mappings>
  <pattern_layouts>
   <layout context="vertical/grouped"
           target="groupbox">
   <feature name="orientation" value="vertical" />
   </layout>
   <layout context="horizontal/ungrouped"
           target="hbox" />
  </pattern_layouts>
</pattern_profile>
```

Listing 5 – Mapping rules definition

To actually generate an application's user interface a generator tool is needed. It combines a pattern enhanced AUI description via above mentioned mapping rules to a CUI for a specific device. Generators need to be written for each destination language; currently we are able to generate XUL and partly Java UI's.

As an example and to demonstrate the results we get from our tools a mail client was modelled and its user interfaces generated for a PC and a PDA. The master_detail pattern was used twice here. Its first application is to select account and folder. The second use is the decision which message from within that folder shall be displayed.

Mapping rules for XUL on PC state that all components can be displayed at once, figure 1 depicts the result: a typical mail application view.

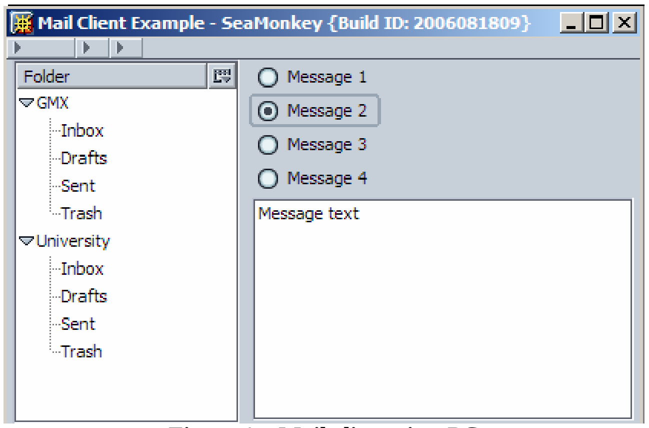

Figure 1 – Mail client view PC

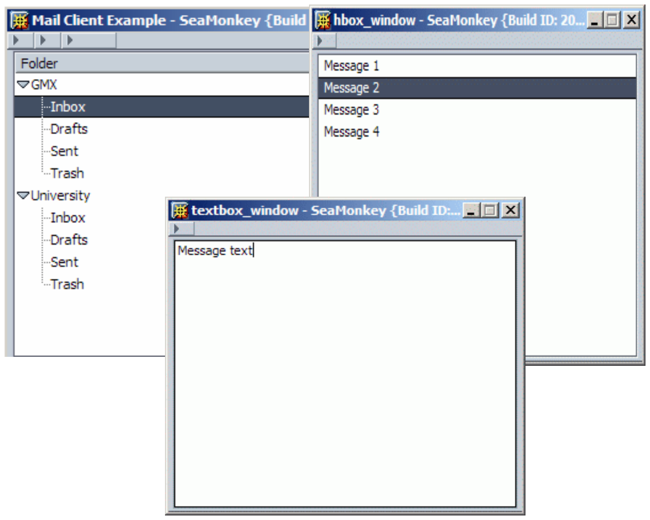
Mail Client Example - SeaMonkey {Build
hbox_window - SeaMonkey {Build ID: 20...
textbox_window - SeaMonkey {Build ID:...
Folder
GMX
Inbox
Drafts
Sent
Trash
University
Inbox
Drafts
Sent
Trash
Message 1
Message 2
Message 3
Message 4
Message text

Figure 2 – Mail client view PDA

Mapping rules and sub-PIC choices for the same AUI on a PDA are declared in a manner to reduce screen space. Each task has its own single view. A user would select folder and account on the first screen (top-left) select the message in screen #2 (top-right) and would get the message's text on a third screen (bottom). Note that the content of the tree, messages selector and message text were added manually for demonstration purposes, as they would be normally provided by an application at run-time.

CONCLUSION

As an effort to make use of the knowledge in HCI pattern catalogues in a model-based UI generation process, an approach was presented of how to integrate such patterns as components into an existing MDA approach.

We proposed to combine pattern catalogues and mapping rules to support varying devices and contexts-of-use, based on the same AUI.

For these purposes enhancements to an existing pattern-language were proposed. An initial toolset supporting a developer in such a process was developed and used to create an exemplary user interface for a mail client.

FUTURE WORK

Our toolset has to be enhanced to generate interfaces in other languages. An advanced Java support seems to be a minimum; it is required to have a better comparison of the potential of our approach in real applications. Secondly, an extensive comparison of our both approaches on pattern instance components is required. Eventually a decision is to be made whether to merge them or drop one. Most important seems the extensions of our pattern catalogue. Currently we only translated five, rather small, patterns. An elaborated investigation similar to [6] should be done.

REFERENCES

1. Wolff, A.; Forbrig, P.; Dittmar, A.; Reichart, D.: Linking GUI Elements to Tasks – Supporting an Evolutionary Design Process, Proc. of. Tamodia 2005, Gdansk, Poland, p. 27-34.

2. Rathsack, R.: Generierung von Gerätespezifikationen aus abstrakten Spezifikationen unter Beachtung von HCI Pattern, Master Thesis, University of Rostock, 2006.

3. PLML: http://www.cs.kent.ac.uk/people/staff/saf/patterns/plml.html

4. Tidwell, Jennifer: Pattern catalogue: http://www.mit.edu/~jtidwell/ interaction_patterns.html

5. Van Welie pattern catalogue: http://www.welie.com/ patterns/index.html

6. Arnout, Karine: From Pattern to Components, PhD dissertation, Swiss Institute of Technology, Zurich 2004

7. 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

8. Müller, Andreas: Spezifikation geräteunabhängiger Benutzerschnittstellen durch Markup-Konzepte, PhD dissertation, University of Rostock, 2003

9. XUL XML User Interface Language: http://www.mozilla.org/projects/xul

10. CTTE: The ConcurTaskTrees Enviroment http://giove.cnuce.cnr.it/ctte.html