Different Kinds of Pattern Support for Interactive Systems

Peter Forbrig, Andreas Wolff
Software-Engineering Group
Institute of Computer Science
University of Rostock
[Peter.Forbrig|Andreas.Wolff]@uni-rostock.de

ABSTRACT
This paper discusses two different approaches for using different kinds of patterns and the corresponding tool support aiming at pattern-supported model transformations. The patterns include GoF design patterns and task patterns.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Language Constructs and Features – *patterns.*

General Terms
Design, Experimentation, Languages

Keywords
Design patterns, Task models, HCI patterns

1. INTRODUCTION
In software engineering object-oriented design patterns, as introduced by the Gang of Four, are considered as valuable support in transferring knowledge. This idea was adapted to the HCI domain. At the beginning patterns for user interfaces were identified and later more abstract HCI patterns like patterns for task models were specified. Patterns are collected in catalogues. Well known examples of pattern-catalogues in the HCI domain were published by Tidwell [9] and Van Welie [11]. Along with these catalogues pattern-languages, as for example PLML [7], evolved to describe each pattern in a standardized manner. However, the term pattern is often used ambiguously. There is no clear definition of patterns and pattern instances. In this paper we try to clarify this point by discussing two tools and their methodology in using patterns. The paper is structured in such a way that a possible tools support for class diagrams and GoF design patterns is discussed in the first section. The second section is devoted to a tool for task patterns. Afterwards strategies for pattern-based reengineering and further research avenues for patterns are discussed.

2. TOOL SUPPORT
2.1 Tools for Class Diagrams

Our first tool supporting GoF design patterns was based on the CASE tool Rational Rose [5]. It used the stereotype mechanism of UML. Patterns were implemented as packages that were classified by the stereotype `<<pattern>>`. In this way it was possible to use the class diagram editor of Rational Rose for specifying classes and associations of patterns. Additionally, cardinalities for classes in side of patterns were introduced. This information is important for the instantiation process of patterns. It specifies the number of possible instances of the corresponding class in the instantiated patters.

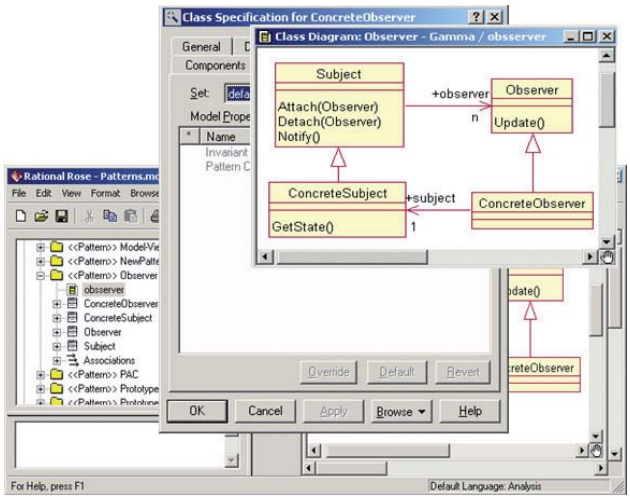

Figure 1. Screen shot of pattern tool based on Rational Rose

In the GoF pattern observer the abstract observer has the cardinality 1, whereas the concrete observer has the cardinality n (see Fig. 1). This means that during the instantiation process several instances of concrete observers can be created. In this way a pattern instance can be created according to the context.

Applying patterns for class diagram transformations has to follow the following sequence of actions:

1. All classes applicable to a transformation of a pattern are selected in a class diagram.
2. A pattern from the library of available patterns is selected.
3. The pattern is instantiated in accordance to the current context of use (selected classes).
4. Classes of the pattern are mapped interactively to selected classes of the class diagram.
5. The transformation of the selected classes to new collections of classes is performed. (Attributes, methods and associations are updated.)

PEICS'10, June 20, 2010, Berlin, Germany

This strategy can be considered as general strategy of the application of patterns. The same idea is implemented in the case tool Together from Borland.

It was the basis for further tools for patterns. Because of lack of space we concentrate on the discussion on a tool that supports the application of patterns to task models.

2.2 Tools for Task Models

The tasks users need to perform to reach a certain goal are identified during task analysis process. Task analysis makes a strong contribution to user-centered design by involving the user perspective of task accomplishment into the design process. Task models that are the result of the analysis process can be the basis for further software development [8] for user interface designer as well as for software engineers. Indeed, they can play a central role for interactive systems.

To reuse knowledge of the task analysis process of other projects the notion of patterns can help. In this way task patterns for login or shopping can be specified.

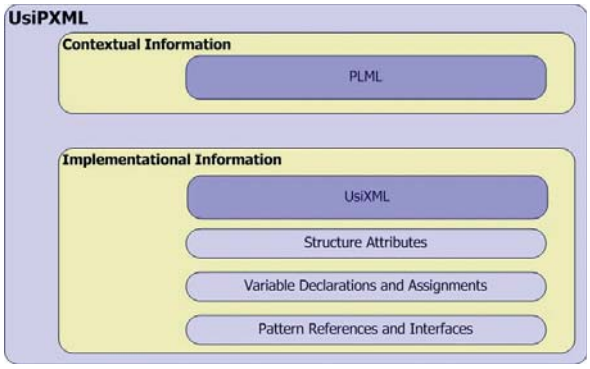

Figure 2. Structure of UsiPXML

UsiPXML (User Interface Pattern Extensible Markup Language) is based on UsiXML and our extension to PLML and is able to specify different kinds of patterns. In this paper we want to focus on task model patterns only.

UsiPXML allows describing contextual and implementational information for a pattern. It is based on PLML [7] and UsiXML [10]. The structure of UsiPXML is illustrated in **Fig. 2**. Its usage is illustrated by the example of the "Login Pattern" that is presented in **Fig. 3**.

On the left hand side of Fig. 3 one can see a part of the contextual information of the "Login Pattern". On the right hand side implementational information is presented. In this case it is the task structure of the login procedure. It consists of the sub-tasks "Show Login Prompt", "Enter Coordinates", "Submit Coordinates" and "Provide Feedback". Below this task structure one can see temporal relations, that express that all above mentioned tasks are in an enabling relation. Structure attributes are introduced to specify whether several instances of the corresponding sub-tree can exist. In this case every task exists only once. This pattern is not generic. In this example there is no need to give different task instances different names. Therefore, variable declarations and assignments do not exist.

However, the pattern uses the "Multi Value Input Form Pattern". While instantiating the "Login Pattern" the "Multi Value Input Form Pattern" has to be instantiated as well. Its implementation is illustrated in **Fig. 4.**

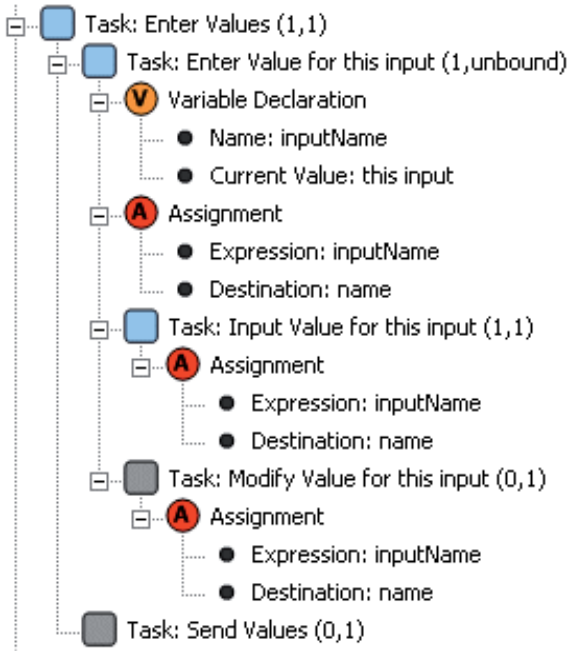

Figure 4. Pattern: Multi Value Input Form

It is specified that an unbound number of values can be entered. It

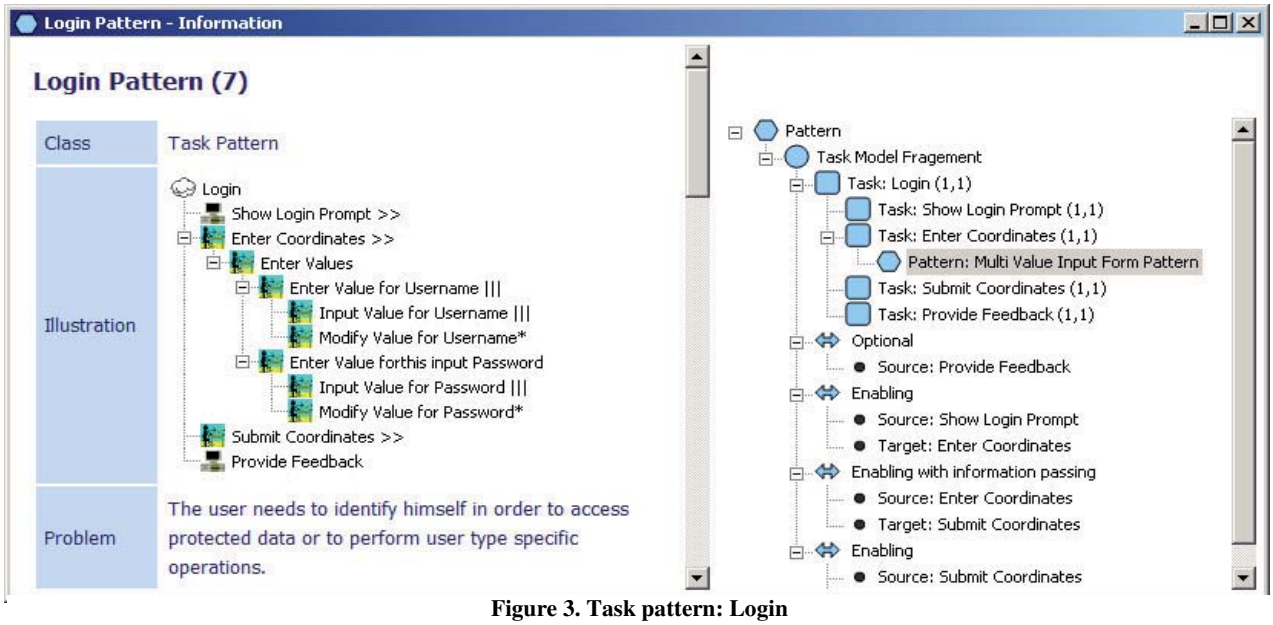

Figure 3. Task pattern: Login

could be interactively decided to instantiate three different inputs and that the first one gets the value "Name", the second one "Number" and the third one "Password". The value of the variables is assigned to different tasks.

As a result the pattern instance of **Fig. 5** is delivered. Based on variable declarations and assignments, generic tasks are specified. The name of the task changes according to the assigned values the names.

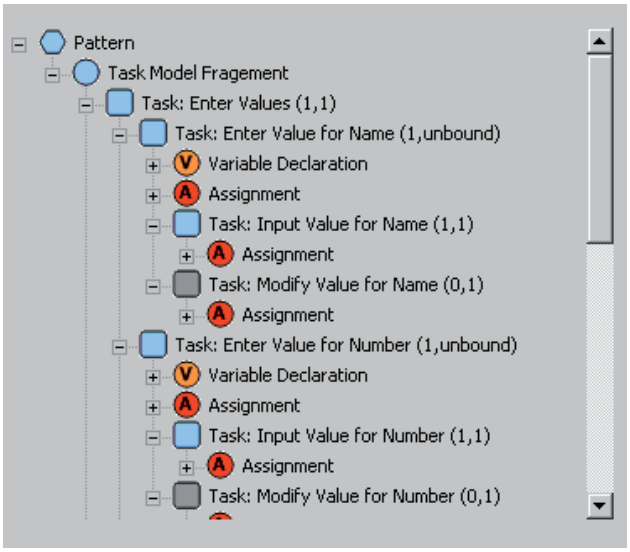

Figure 5. Instantiated Pattern: Multi Value Input Form

At the moment the transformation of task models by task patterns is simplified in such a way that only leaves of the task tree can be replaced by pattern instances. It might be useful to be able to replace sub-trees by transformed sub-trees.

To support such an approach a mapping concept of corresponding elements in the model and the elements pattern instance would be necessary.

3. DISCUSSIONS

Arnout [1] classified design patterns according to their ability to be implemented as components. This classification is related to possible tool support. Indeed, some patterns have such level of abstraction that nearly no tool support is possible. The same is true for HCI patterns. Fortunately, some patterns can be used to transform models in an interactive way. To apply a pattern the following procedure has to be followed

1. Identify a sub model S as a part of a model M.
2. Identify a pattern P that is applicable to the sub model S.
3. Instantiate pattern P to the pattern instance PI according to the sub model S.
4. Map elements of PI to elements of S.
5. Transform S according to PI delivering the result SP.

At the moment some tools lack the ability to provide the history of pattern applications. However, this feature is very important for reengineering of existing systems based on a different pattern set. This is important if a pattern can be implemented on different platforms by different patterns.

In [14] we presented an approach that allows the reengineering of software based on patterns. **Fig. 6** gives an overview of the system. Based on the existing software an abstract user interface is reverse engineered. This abstract user interface is the basis for a forward engineering process. All transformations are based on patterns. At the moment XUL [16] plays an important role for this tool, but it can be easily replaced by any other XML dialect like UsiXML [10].

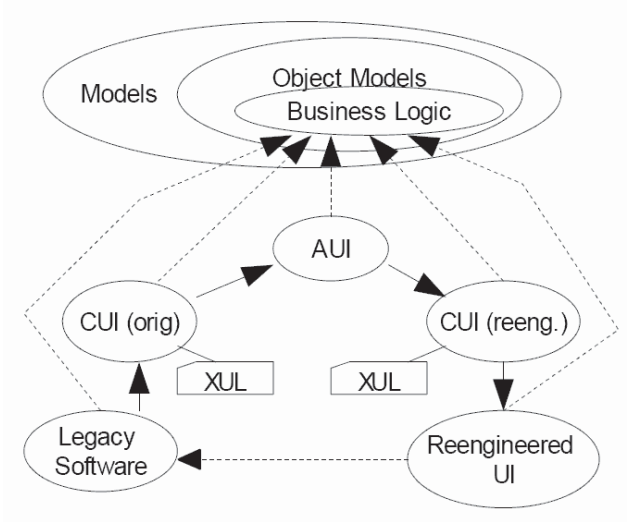

Figure 6. Pattern-based reengineering approach

Patterns that can be implemented as a component can be considered as building blocks for the development of interactive software. At the moment patterns are always related to a specific kind of models. GoF design patterns are related to object models, Task patterns are related to task models and user interface patterns are related to user interface models.

In the future it might be helpful, to identify and specify patterns (or let us say generic components) that overcome the borderline between different models. There could be a "Login Patterns" that consists of a task model (like in our example above), but it comes already with an object model, a dialog model and a user interface model.

It would be nice if this idea could be discussed during the PEICS workshop.

Assuming, that a set of such patterns exist, new challenges appear because different levels of reuse can be considered. For example, there could be a software that already has a user interface specified in detail. In this case, the concrete user interface of a pattern has to be exchanged with a new one because the old user interface of the pattern might not fit to the already specified software. Different level of abstraction of the user interface model can help. If the user interface of the pattern is structured hierarchically the appropriate level of abstraction for reuse has to be selected. This seems to be a difficult problem as well.

At the moment, our tool for task patterns can inject the pattern instance into our notation of task models or export the pattern instance as UsiXML file. The tool runs within the eclipse environment [4]. Maybe UsiXML and eclipse can be a common basis for different tools in the future. In this way different ideas and approaches could be combined. A collection of cooperating tools could be developed at different locations. The identification

of a common platform and a common framework could be a further result of the workshop.

4. SUMMARY AND OUTLOOK

In this paper we discussed tool support for the application of patterns during the development of interactive systems. We concentrated on two typical tools that are designed for GoF patterns and task patterns respectively.

It was shown how specific patterns, that can be specified by some kind of components, can be interactively used to transform and improve models.

The challenge of the future will be to combine the different approaches and to identify more complex patterns that consist of parts of different models. Tool support for such kind of patterns will be a challenge as well.

5. REFERENCES

[1] Arnout, Karine: From Pattern to Components, PhD dissertation, Swiss Institute of Technology, Zurich 2004

[2] Ding CTTE: The ConcurTaskTrees Enviroment http://giove.cnuce.cnr.it/ctte.html

[3] van Duyne, D., J. Landay, and J. Hong, The Design of Sites - Patterns, Principles and Processes for Crafting a Customer-centered Web Experience, Boston, USA: Addison Wesley. (2005)

[4] Eclipse environment: http://www.eclipse.org/

[5] Forbrig, P.; Lämmel, R.; Mannhaupt, D.: Pattern-Oriented Development with Rational Rose, Rational Edge, Vol. 11 No. 1, 2001.

[6] Gamma, E., et al., Design Patterns : Elements of Reusable Object-Oriented Software. Addison-Wesley professional computing series. 1995, Reading, Mass.: Addison-Wesley.

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

[8] Schmidt, B. and Riss, U. V. 2009. Task Patterns as Means to Experience Sharing. In Proceedings of the 8th international Conference on Advances in Web Based Learning (Aachen, Germany, August 19 - 21, 2009). M. Spaniol, Q. Li, R. Klamma, and R. W. Lau, Eds. Lecture Notes In Computer Science, vol. 5686. Springer-Verlag, Berlin, Heidelberg, 353-362.

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

[10] UsiXML: http://www.usixml.org

[11] Van Welie pattern catalogue: http://www.welie.com/ patterns/index.html

[12] 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

[13] 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.

[14] Wolff, A.; Forbrig, P.: Model-based Reengineering of User Interfaces, MDDAUI Workshop organized October 1, 2007 at MoDELS'07, Nashville, Tennessee, USA

[15] Wolff, A.; Forbrig, P.: Deriving User Interfaces from Task Models, MDDAUI Workshop organized at International Conference on Intelligent User Interfaces (IUI 2009), Sunday, February 8th, 2009, Sanibel Island, Florida, USA

[16] XUL XML User Interface Language: http://www.mozilla.org/projects/xul