Patterns, Tools and Models for Interaction Design

Daniel Sinnig[1, 2], Ashraf Gaffar[2], Ahmed Seffah[2] and Peter Forbrig[1]

(1) Software Engineering Group
Department of Computer Science,
University of Rostock,
18051 Rostock, Germany
pforbrig@informatik.uni-rostock.de

(2) Human Centered Software Engineering Group
Department of Computer Science,
Concordia University,
1455 De Maisonneuve Blvd. West,
H3G 1M8, Montreal, Canada
{d_sinnig, gaffar, seffah}@cs.concordia.ca

ABSTRACT

In recent years the re-use of already existing solutions and ideas has become more and more crucial. Re-inventing the wheel over and over again is not feasible. Especially model based development approaches suffer from the lack of libraries populated with existing solutions and ideas which must just be tweaked in order to applicable to different context of use. Patterns have the potential to overcome this major shortcoming. In this paper we will try to stimulate and foster to idea of transferring the idea of patterns to the model-based development community. We will introduce patters as medium to capture ideas and solutions within the domain of model based design.

Moreover the lack of tool support has led to situation that model – based approaches have not been fully acknowledge by the developers of interactive applications. In recent years a set of tools has been developed in our group, which the developer with the establishment of the various models. Therefore In the following we will introduce our model – based philosophy and the possible application and impact of patterns for each of our models. Moreover we will briefly introduce some of our tools and ideas which might be helpful with the creation of the various models.

Keywords

Patterns, re-use, interactive applications, task, dialog, presentation, layout

INTRODUCTION ON PATTERNS

The architect Christopher Alexander introduced the idea of patterns in the early 1970s [1, 2]. He introduced patterns as a three-part rule to help architects and engineers with the design of buildings, towns, and other urban structures. His definition of a pattern was as follows: "Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution".

The concept of patterns has been transferred to the software community by [4]. Their book "Design Patterns" contained a collection of patterns for the design of object–oriented software. The book has been widely acknowledged and referenced within the community.

Recently, like in the software engineering community, the user interface design community has also been a forum for vigorous discussions on pattern languages for user interface design and usability engineering. UI patterns are an effective way to transmit experience about recurrent problems in the HCI domain related to UI design issues. A pattern is a named, reusable solution to a recurrent problem in a particular context of use. In other words UI patterns capture the essence of successful solutions to recurring design problems. Correctly applied they ease and accelerate the development of initial prototypes and assist with design choices. A pattern takes what was previously an art of designing usable software and turns it into a reusable unit.

Until now, patterns have been used mainly used as tools for designers in the same sense as UI development has been treated as a rather creative design activity. However, with the advent of pervasive computing and mobile users the design and the development of UIs has become increasingly complex. Thus, UIs must be aware of dynamically changing contexts and withstand variations of the environment. From this emerges the need for a structured engineering-like development approach. Model based approaches have the potential to establish the basic foundation for a systematic engineering methodology for UI development. Thus, also the idea of patterns should be shifted from design to systematic development.

In the next section we will outline our model – based development philosophy for interactive applications. We will discuss the use of patterns for each model. Moreover we will introduce some of our tools.

OUR MODEL-BASED PHILOSOPHY

In a model based UI design methodology for interactive applications various models are used to describe the relevant aspect of the User Interface. Many facets exist as well as related models. As depicted in Figure 1 our approach consists of: task models, user models, business object models, a dialog model, a presentation and a layout model.

Our approach starts of - like a typical development lifecycle - and begins with domain analysis. To date the creation of the task model has been commonly agreed to be a reasonable starting point [10]. Therefore our approach starts with the elicitation of the user's tasks, as they are currently performed, resulting in the task model. As a consequence this task model is also often named "existing" task model. Additionally models for capturing user characteristics and business objects are developed. All three analysis models are portrayed by the light shaded ellipse at the top of Figure 1. In addition it is shown that the task model has to be modeled in mutual relationship to the user model, representing the functional roles users have to play for task accomplishment, as well as their individual perception of the tasks. The user model is also related to the business-object model and the task model since the user may require different views on the data while performing a task. Besides, the task model has to reflect the abilities, skills, and preferences of end users. A relationship between the business-object model and the interaction model is required, since the problem domain data in the business-object model has to be presented to the end users for interactive task accomplishment.

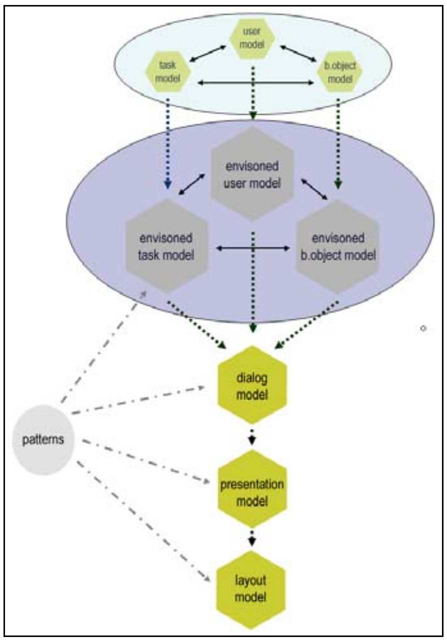

Figure 1 Model based development at a glance

The introduction of a new interactive application leads inevitably to a change of the role of the user and the tasks, which the user will perform. Furthermore also new interactive tasks (the actions the user performs with the new interactive system) must be modeled. Thus, after domain analysis first design decisions are made by establishing the envisioned task, user and object models with consideration of the future interactive system. The dark shaded ellipse of Figure 1 illustrates that these modes are mutually related as well.

Model – based design focuses on finding mapping between the various models. [11] Thus, at this point based on these rather abstract task, user and object models, a dialog, a presentation and a layout model are derived to reveal some implementation details of the user interface.

In a next step the dialog model is developed. It specifies the navigational structure of the UI and the interaction techniques. [11] It is a more specific model and can be derived in good part from the more abstract task-, user- and business object models.

From the dialog model the presentation and layout model are derived. The presentation model maps the elements of the different dialog views (defined in the dialog model) to abstract interaction objects, such as menubar, groupbox, listbox, etc. The layout model defines the arrangements and the style of the user interface objects according to an overall floor plan.

After outlining the basic structure of our approach, we will now discuss which patterns can impact the establishment of the task, dialog, presentation and layout model.

TOOLS AND PATTERNS FOR MODELS

In our approach we are aiming to use patterns as building blocks in order to create these models. Different kinds of models require different kinds of patterns which must be formulated in a different way.

Task Model

The task models can be impacted by so called task patterns. Task patterns have been first introduced by Paterno [3, 7] as reusable structures for task models. The patterns were described as hierarchical structured task fragments which can be used to successively build the task model. Based on this idea we have developed the tool "Task Pattern Wizard". The program is able to read and visualize already existing task descriptions that are specified in XIML [14]. It is also capable to interpret task patterns descriptions documented in a prototypical XML based mark-up language. After the task pattern has been instantiated the resulting task fragment can be inserted resulting in a modified task model.

In contrast to Paterno's task patterns, which are defined as fixed fragments, the Task Pattern Wizard expects the patterns to be specified in a dynamic fashion. Variables are used as placeholders for the particular context of use. During the so called process of pattern adaptation the variables are replaced by concrete values representing the current context of use. Different kinds of variables exist. Among them: "Substitution variables" and "process variables". On the one hand substitution variables are simply used as placeholders for certain values. During the process of pattern adaptation the Task Pattern Wizard will question the user for entering values for these variables. Then, in a top down process each occurrence of a substitution variable will be replace (substituted) with this value. On the other hand, process variables are used to describe the structure of the task fragment, which will be created by the pattern. For example entering values into a form is very repetitive. The same basic task (enter a value) appears over and over again. Basically each peer task can just be distinguished by its name and the kind of input. Thus, instead of task description for each of these peer tasks, a process variables, that signals the number of respective tasks, can be used. A more detailed descriptions about task patterns and the process of pattern adaptation can be found in [8, 9]

This generic notation of patterns ensures that the same pattern is applicable to different contexts of use. The past has shown, that many approaches of reusing knowledge failed, because they have been designed too specific and rigid. (too context sensitive and domain dependent). Once these knowledge fragments were ordered and classified in a way, that efficient usage was possible, most of them were not applicable anymore since the context has slightly changed. Therefore we believe that the solution stated in the pattern should be formulized generic enough in order to withstand variations of context and domain.

In order to assess and validate the correctness and appropriateness of the task model we have developed the XIML Task Simulator [5]. It presents different views on the tasks, their users and used (tools) and modified objects (artefacts). Moreover tasks can be animated and the user can step through possible task scenarios (Pluralistic walkthrough) [6] within the scope of the underlying task model.

Dialog Model
After developing the task model the dialog model is interactively derived from the task, user and object model. The dialog model associates several tasks to dialog views and defines transitions between these dialog views. At this stage dialog patterns can help grouping the tasks and suggest sequences between dialog views.

Finding dialog views and transitions is closely connected to the underlying task models. On the one hand, structural information from the task model, which describes the task–subtask hierarchy, can be used to group related tasks into task views. On the other hand temporal transitions between sub tasks can be used to constrain and derive possible dialog transitions [11, 7]. Consequently patterns applied to the task model indirectly affect the dialog model and in particular the dialog graph. An example of a dialog pattern is the wizard pattern (adopted from the wizard pattern by Welie [13]. It describes a sequential run through a number of dialogs until an end dialog has been reached. We are currently experimenting to formulate such patterns with XIML.

We have already developed the tool “Dialog Graph Simulator” [5]. It allows to group different task to dialog views and the definition of transition between various dialog views. The designed dialog structure can be further saved to XIML format and thus re-processed by other tools. Moreover a first abstract prototype of the interface can automatically be generated out the dialog description. Using the cognitive walkthrough method [6] users can walk through the interface in order to accomplish predefined tasks. Whenever the interface blocks the user from completing a task, it is an indication that the interface or the underlying task description is missing something. For the future we envision that the Dialog Graph Simulator can also process dialog patterns and thus semi-automatically establish transitions between the various dialog views.

Presentation and Layout Model
Next, in order to develop the presentation model the tasks of each dialog view are associated with interaction elements such as buttons, trees and lists. Moreover, some domain objects (tools or artefacts) which are related to the tasks are also mapped to interaction elements. Presentation patterns can be applied in order to map complex tasks (such as advanced search) to a predefined set of interaction elements. We are currently experimenting to describe such presentation patterns as Velocity XUL templates [12, 15]. Presentation patterns describe fragments of the presentation model. Each fragment describes one or a set of interaction objects.

Finally the interaction objects are positioned following an overall layout or floor plan described in the layout model. Additionally, the visual appearance of each interaction element is specified by setting fonts, colors and dimensions. In our framework layout patterns -which are described as XUL templates as well- are used to integrate proven layouts and design solutions. An example of such a layout pattern is the “3 – Column Layout” introduced by Welie [13]. Practically the loose set of XUL fragments of the presentation model is aggregated to XUL code. Finally this XUL code is automatically rendered to a concrete user interface implementation.

According to our model – based framework the presentation model and layout model are logically separated. In many model-based development approaches these models are summarized to one model. However we decided to split them up, since we believe that for each model different kinds of patterns apply. On the one hand patterns that describe a set of interaction elements (presentation patterns) and on the other hand patterns that describe the layout of the interaction elements (layout patterns).

DISCUSSION AND OPEN QUESTIONS
In this paper we have introduced our model based framework and have outlined the application of patterns and the use of various tools. It was shown that patterns as a medium for capturing and disseminating knowledge are also an interesting tool in the domain of model – based design. First tools and approaches for integrating patterns into the development framework have been introduced. However the work presented in this paper is of preliminary nature, where the following questions remain open:

Existing patterns catalogues embody a substantial amount of knowledge. How to make efficiently use of this knowledge within the domain of model based design.

Different kinds of patterns exist as well as model. Which patterns are useful for which model?

Which models would benefit most from the usage of patterns?

Do patterns really speed up or improve the development of interactive applications?

Is it feasible to create a universal pattern catalogue which applies to users of different background pursuing different goals?

What can the usage of patterns be validated?

REFERENCES

1. Alexander, C. *The Timeless Way of Building*. New York: Oxford University Press, 1979.
2. Alexander C., Ishikawa S., Silverstein M., Jacobson M., Fiksdahl-King I., and Angel S., *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press, 1977.
3. Breedvelt, I., Paternò, F. & Severiins, C., "Reusable Structures in Task Models", *Proceedings Design, Specification, Verification of Interactive Systems '97*, Granada, June 1997, Springer Verlag, pp.251-265.
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns: Elements of Object-Oriented Software*. Book published by Addison-Wesley, 1995.
5. Forbrig, P., A. Dittmar, D. Reichart, and D. Sinnig, "User-Centred Design and Abstract Prototypes", *in Proceedigns of BIR 2003*, (Berlin), pp. 132 - 145, SHAKER, Sept. 2003.
6. Nielsen, J. ; Mack, R.: *Usability Inspection Methods*, Book published by John Wiley & Sons, New York, NY, 1994, ISBN 0-471-01877-5.
7. Paternó, F. *Model-Based Design and Evaluation of Interactive Applications*, Springer, 2000.
8. Sinnig, D., H. Javahery, P. Forbrig, and A. Seffah, "The Complicity of Model-Based Approaches and Patterns for UI Engineering", in Proceedings of BIR 2003, (Berlin), pp. 120 - 131, SHAKER, Sept. 2003.
9. Sinnig. D., Gaffar, A., Seffah, A., Forbrig, P., "Patterns in Model-Based Engineering, Proceedings of CADUI 04, Portugal, 2004.
10. Vanderdonckt, J., Puerta, A.: Introduction to Computer-Aided Design of User Interfaces, *Proceedings of the CADUI'99*, Louvain-la-Neuve, Kluwer Academic Publishers, 1999.
11. Vanderdonckt, J., Limbourg, Q., Florins, M., Deriving the Navigational Structure of *a User Interface, In Proceedings of INTERACT 2003*, Sept. 2003, Zuerich, IOS Press, pp.455-462.
12. Velocity, *Velocity*, http://jakarta.apache.org/velocity/, 2003.
13. Welie, M., *Patterns in Interaction Design*, http://www.welie.com, 2003.
14. XIML, *eXtensible Interface Markup Language*, http://www.ximl.org, 2003.
15. XUL, *XUL Planet*, http://www.xulplanet.com/, 2003.