Chapter 16

PATTERNS IN MODEL-BASED ENGINEERING

Daniel Sinnig[1,2], Ashraf Gaffar[2], Daniel Reichart[1], Peter Forbrig[1]
and Ahmed Seffah[2]
[1]*Software Engineering Group, Department of Computer Science,*
University of Rostock, Rostock (Germany)
E-mail: {daniel.reichart, Peter.Forbrig}@informatik.uni-rostock.de
[2]*Human-Computer Software Engineering Group, Department of Computer Science,*
Concordia University, Montreal (Canada)
E-mail: {d_sinnig, gaffar, seffah}@cs.concordia.ca

Abstract In this paper we demonstrate how patterns can act as a driving force for the development of interactive applications. As knowledge re-use is becoming more and more crucial, patterns can be an effective tool to represent knowledge of the HCI domain. Using a model–based development methodology, it is shown how patterns can act as building blocks for the establishment of these models. Starting from outlining the general process of pattern application, we discuss how and which patterns are suitable for several models. In particular we discuss the application and use of patterns for the task, dialog and presentation models. Furthermore, we suggest an interface for patterns using "generic classes" and give concrete examples to corroborate our approach. This allows for modular patterns reuse and plausible parameter exchange with the underlying system. Tool support is based on XML-representations of patterns using a template engine.

Keywords: Model-based interface design, Patterns, Task modelling, UI engineering.

1. PATTERNS FOR MODEL-BASED DESIGN

The concept of patterns has been transferred to the software community by [5]. 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 (UI) 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. Even though patterns serve a number of different purposes (education, discussion ground, re-use, etc.), this paper mainly looks at patterns as vehicles for re-use of existing solution.

Since UI patterns capture the essence of successful solutions to recurring design problems, correctly applying them will help avoid "re-inventing the wheel". They could accelerate the development of initial prototypes and help designers reuse successful, elegant designs without the need to rediscover these designs [5]. In the following, we will introduce our approach of developing a formal notation for patterns within the scope of a model–based design of interactive applications.

1.1 The Impact of Patterns on the Model–Based Framework

In a model based UI design methodology, various models are used to describe the relevant aspects of the user interface. Fig. 1 portrays that many UI facets exist and reflects the relevant models for tasks, business objects (domain), users, dialogue and presentation.

First, design decisions are made to establish the envisioned task model in which the future support of the interactive system is already considered. Additionally, models for capturing user characteristics and business objects are developed. Based on these rather abstract models, a dialog, a presentation and a layout model are derived to reveal some implementation details of the user interface. Due to the lack of tool support and libraries populated with existing solutions and ideas, model based user interface design has not reached the mainstream software developer, yet [14]. We believe that patterns have the potential to overcome this major shortcoming. Therefore, as demonstrated in Fig. 1, in our approach we are aiming to use patterns as building blocks in order to create the various models. In the following we will demonstrate how patterns can impact several models. In particular we will focus on the impact of patterns on the envisioned task model, the dialog model, the presentation model and the layout model.

1.2 The Process of Pattern Application

In the domain of software development, the reuse of ideas and knowledge is becoming more and more crucial as a solution to the stark competition, the demand on more quality and less time-to-market, and the steady increase in complexity [10]. "Reusability" is considered as an important quality factor

[7]. Using patterns can be an effective way to transmit experiences about recurrent problems in the software and UI development domain. Therefore a solution should be generic enough to apply to different contexts of use. In other words patterns should be formulated generically enough to withstand variations of context and domain. Before the pattern solution stated in the pattern is really tangible and applicable, it must be adapted to the current context of use.

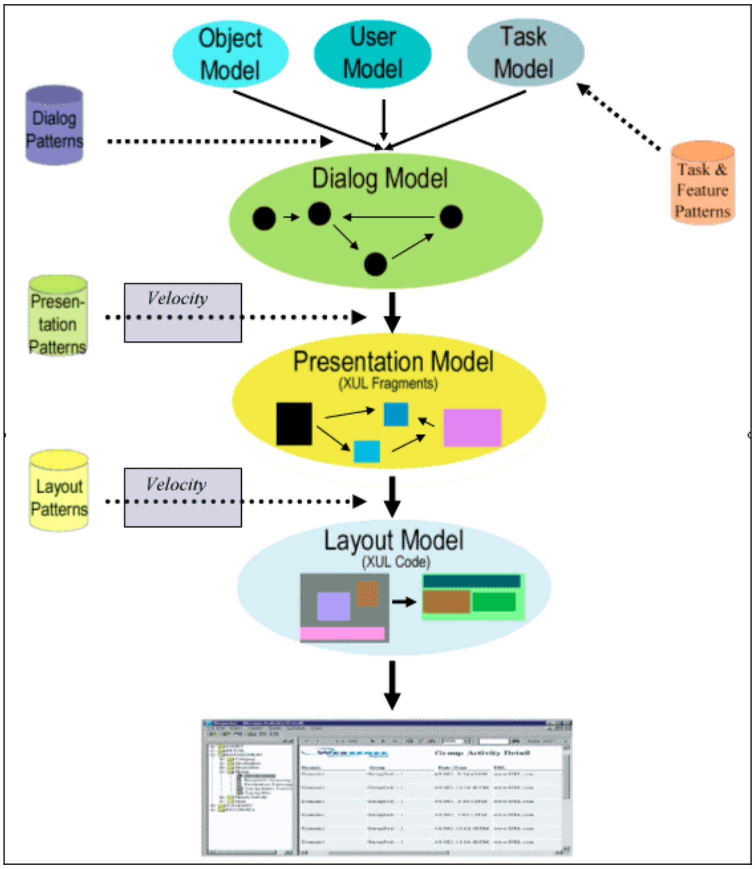


Figure 1. Patterns as building blocks within a model based methodology.

Thus we suggest that patterns contain variables, which can act as placeholders for each particular context of use. In other words, the variables must be bound to concrete values representing the surrounding context. Assuming that patterns are applied to models, the process of pattern applications comprises four sequential steps:

1. **Identification:** A subset M' of the target model M is identified: $M' \subseteq M$. This should reduce the domain size, and help focus the attention on a smaller subset of concern for the next step.

2. **Selection:** An appropriate pattern *P* is selected to be applied to *M'*. By focusing on a subset of the domain, the designer can scan *M'* more effectively to capture potential "spots" that could be improved using some patterns. This is the most important step of all the four. It depends strongly on the experience and the creativity of the designer.
3. **Adaptation:** A pattern is an abstraction that must be instantiated. Therefore in this step the pattern *P* will be adjusted according to the context of use resulting in the pattern instance *S*. In a top down process all variable parts will be bound to specific values, resulting in a concrete instance of the pattern.
4. **Integration:** The pattern instance *S* will be integrated into *M'* by connecting it to the other elements in the domain. This may require replacing, updating or otherwise modifying other objects to produce a seamless piece of design.

Automatic tool support is important in order to integrate patterns effectively into the development life cycle of interactive applications. Moreover by integrating the idea of patterns into development tools, patterns can be a driving force throughout the entire UI development process. For instance the top-down process of pattern adaptation can be greatly assisted by tools such as Wizards. A Wizard runs through the pattern tree and questions the user each time it encounters a variable that has not been resolved yet. We have developed a prototype of a task pattern wizard (introduced in [11]), which supports all phases of pattern integration for the task model, ranging from pattern selection over pattern adaptation until pattern integration. Which patterns are applicable for the task model is introduced in the next section.

2. PATTERNS FOR THE ENVISIONED TASK MODEL

The envisioned task model describes how activities can be performed to reach the user's goals when interacting with an interactive system [9]. Using task models, designers can develop integrated descriptions of the system from a functional and interactive point of view. Task models typically are hierarchical decompositions of goals, tasks and subtasks into atomic actions [12]. Also the relationships between tasks are described in correlation with the execution order or dependencies between peer tasks. The tasks may contain attributes about the importance, the duration of execution and the frequency of use. In order to speed up the process of establishing the task model and to integrate proven and efficient solution, we suggest using patterns as building blocks. In the following we will explain how patterns for the task model should be written and how they should be applied. In a subtle manner we distinguish between two kinds of patterns that are applicable for the user-task model: Task Patterns and Feature Patterns.

- **Task Patterns** describe the activities the user has to perform while pursuing a certain goal. The goal description acts as an unambiguous identification for the pattern. In order to compose the pattern as generic and flexible as possible, the goal description should entail at least one variable component. As the variable part of the goal description changes, the content solution part of the pattern will adapt and change accordingly. Task Patterns can be composed out of sub-patterns. These sub-patterns can either be task-patterns or feature-patterns.
- **Feature Patterns**, applied to the user-task model describe the activities the user has to perform using a particular feature of the system. For the purpose of this paper we define a feature as an aid or a "tool" the user can use in order to fulfil a task. Examples of these features can be "Keyword Search", "Login" or "Registration". Feature patterns are identified by the feature description, which should also contain a variable part, to which the realization of the feature (stated in the pattern) will adapt. Feature patterns can comprise other sub-feature patterns.

As we mentioned above, the difference between task and feature patterns is subtle, but noticeable. While task patterns concentrate on a specific goal, the same task can be accomplished in different ways using different feature patterns. That is why feature patterns are important as a classification. Similarly, the same feature pattern can be used to accomplish different task patterns. Therefore it is safe to say that there is a many-to-many relationship between the two. To summarise, Task patterns are concerned with the user goals (what we need to do), while feature Patterns are concerned with the system behaviour (how we can do it). A typical task performed in many different applications is to find something. This can range from finding a book at www.amazon.com to finding a used car at www.cars.com, to even finding a computer in a network environment. All these tasks embody the same basic task and can just be distinguished by the particular "Find" object in the goal description. In order to create a generalised *Find* Pattern, we must abstract the particular object we are searching, and replace it with a generic variable.

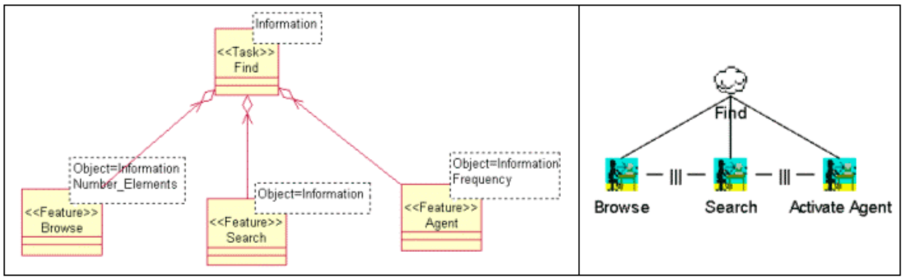


Figure 2. Pattern for find information.

For the sake of simplicity, let us assume that a simplified version of the *Find* pattern suggests that "Find" information can be performed by browsing, searching or using an agent. In the left part of Fig. 2 the *Find* pattern is displayed in an abstract manner. We have used the UML notation for parametric classes is used to portray the pattern. The variable "Information" is utilised as a placeholder for the particular type of information one is trying to find. In the right part of Fig. 2 a possible instance of the pattern is shown. The details of the resulting task tree are illustrated with CTTE notation [9].

Moreover, it is visualised that the *Find* pattern is composed of the feature patterns *Browse*, *Search* and *Agent*. If we place patterns in such an "aggregation" relation we have to pay special attention to the variables. It is shown in Fig. 3 that a variable, defined at the super-pattern level can affect the variables used in the sub patterns. The value of the variable "Information" of the *Find* pattern is used to assign the "Object" variable in all sub patterns. However the variables "Number_Elements", and "Frequency" of the sub-patterns *Browse* and *Agent* remain undefined. During the process of adaptation, the variables of each pattern must be resolved top-down and replaced by concrete values.

In Fig. 3, we have bound the variable "Information" with the value "Book" to create the patterns instance *Find Book*; and with the value "Car" to create the instance *Find Car*. Please note that with the binding of a concrete value to the variable "Information" in the goal description, the body of the pattern has changed accordingly. After the pattern adaptation process, the patterns instance can be integrated in an already existing task model. In Fig. 3, *Find Car* has been integrated into the Car-shop task model. This process of integrations is visualised using the inheritance relationship and can be interpreted as: Car-shop has inherited all methods (tasks) from *Find Car*. Eventually after resolving all variables, the pattern instance will be transformed into a concrete task structure. Practically this integration process is not realised by inheritance. It is supported by a wizard, which is described in [11].

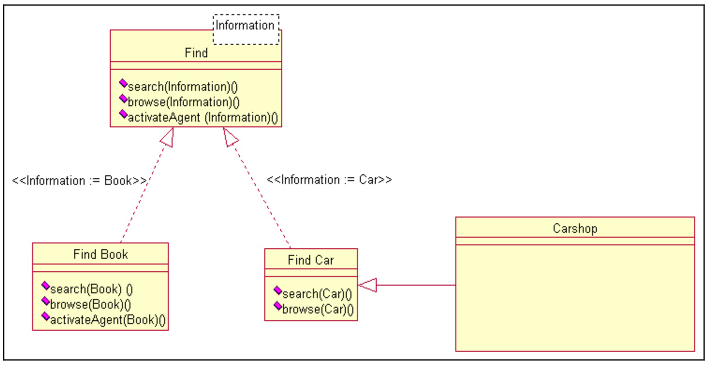


Figure 3. The Find pattern and its instances.

3. PATTERNS IN THE PROCESS OF DERIVING THE UI FROM ABSTRACT DESCRIPTIONS

Until now it was described how patterns can be used as building blocks for establishing task models. Task models as well as user and object models are rather abstract and deal only indirectly with user interface issues. In the following we will explain how an implementation of a user interface can be derived from these abstract descriptions. Moreover it will be shown how patterns can drive and influence this process.

In Fig. 1, we have portrayed four milestones on the way from an abstract description to the implemented user interface. First a 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. 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.

Within our approach presentation patterns are described as Velocity XUL templates [18]. Thus, our presentation model consists of a set of XUL code fragments. Each fragment describes one or a set of interaction objects. After that the interaction objects are positioned following an overall layout or floor plan resulting in the layout model. Additionally, the visual appearance of each interaction element is specified by setting fonts, colours and dimensions. In our framework layout patterns –which are described as XUL templates as well– are used to integrate proven layouts and design solutions. 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. In the following we will explain in greater detail each phase.

According to our model–based framework the presentation model and layout model are logically separated. We decided to split them into two categories, since we believe that for each model different kinds of patterns apply. The first category contains patterns that describe a set of interaction elements (presentation patterns). The other category contains patterns that describe the layout of the interaction elements (layout patterns).

3.1 Patterns and the Dialog Model

The dialog model specifies the user commands, interaction techniques, interface responses and command sequences that the interface allows during user sessions. It must encompass all static and dynamic information the user needs for the dialog with the machine. This information is grouped into several dialog views. The dialog view contains functionally- and logically related elements of the task model and the business object (domain) model.

In short, the dialog model specifies the navigational structure of the UI and the interaction techniques [15]. It is a more specific model and can be derived in good part from the more abstract task-, user- and business object models.

There are different strategies to design the dialog model. One possibility is the evolution of the task model to a final user interface. Janus [1] uses information mainly from the object model. However, most approaches are based on tasks. TERESA [6] follows an idea of grouping tasks based on preconditions, which allows an automatic generation of dialogue models.

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 [9, 15]. Consequently patterns applied to the task model indirectly affect the dialog model and in particular the dialog graph. Let us take the example of the *Multi Value Input Task* Pattern introduced by [2]. For the sake of simplicity let us assume that our *Multi Value Input Task* pattern describes a task structure in which the user edits various values. After all values have been entered the user can submit them.

In the left fraction of Fig. 4 the interface of the dialog pattern *Wizard* is illustrated. It is parameterised with the variable "Number" which stands for the number of dialogs the wizard will run through. Let us assume that we will use the *Wizard* pattern in order to realise the *Multi Value Input Task.* An instantiation for three input tasks of the pattern is depicted in the right fraction of Fig. 4 and visualised as a dialog graph using the notation introduced by [4].

Adopting the semantics of this graph, the user sequentially inputs three values. After entering the third value submit can be performed and the dialog view will be closed. In particular the user runs through a sequence of three single dialogs, starting from dialog one. From each dialog only one of the neighbour dialogs (previous or next) can be reached. After submitting the third dialog the overall dialog view will be closed.

In order to validate, find and apply dialog patterns we have developed a tool called "DialogGraphSimulator" [4]. Using our application the user can interactively "walk through" the dialog graph. The DialogGraphSimulator helps to define multiple dialog structures for one task model. Different attempts can be opposed to each other and the best solution can eventually be extracted. Due to the separation of task and dialog structure, dialog patterns can be brought in independently. As the DialogGraphSimulator processes dialog structures described in XIML [3], our dialog patterns are formalised as XIML fragments as well. Currently we are developing a tool, similar to the TaskPatternWizard, to allow the computer aided adaptation and integration of dialog patterns.

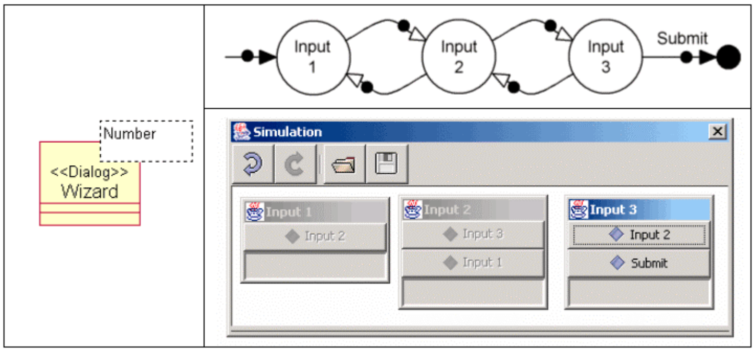


Figure 4. Interface, instance and simulation of the Wizard dialog pattern.

Fig. 4 (lower right corner) contains also a screenshot of the simulation of the *Multi Value Input Task* pattern implemented as a *Wizard*. Each dialog is visualised by a little window. Buttons in each window portray the possible transitions to other dialog views. The third input dialog is currently active. From this dialog it is either possible to go back to the second dialog or to press submit, which would close the dialog view.

3.2 Patterns and the Presentation Model

In the presentation model, a set of abstract UI elements is defined to determine the abstract appearance of the user interface. In particular, the grouped tasks of each dialog view are associated with a set of interaction elements such as buttons, trees and lists. Moreover some domain objects, which will be displayed on the interface, are mapped to interaction elements as well. Please note that all interaction elements should be described in an abstract manner. Style attributes such as size, font, and color remain open and will be defined by the layout model.

We have chosen the generic user interface description language XUL [18] as a medium to describe the presentation model. Thus our presentation model basically consists of a set of XUL fragments. Each fragment represents a single interaction elements or a group of interaction objects.

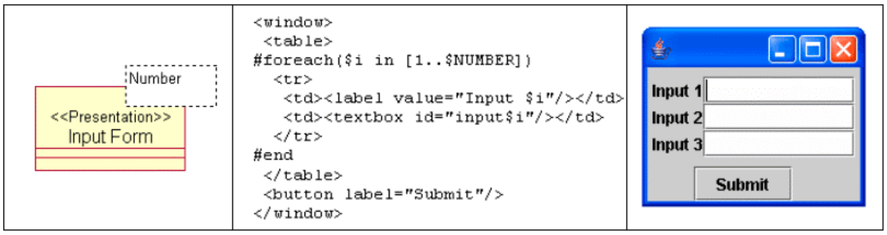
For the presentation model, presentation patterns embody building interface object blocks. In practice, instantiations of presentation patterns deliver

XUL fragments which -again- describe user interface objects. Therefore each presentation pattern must have a mechanism to generate variants of XUL code depending on the assignment of their variables. We have chosen to employ XUL Velocity templates in order to implement the patterns. Velocity templates can be used to dynamically generate XUL code. If the variable parts of the presentation pattern change, conditions, loops and other control structures are used to adapt the template (pattern) accordingly.

Fig. 5 portrays three different views on a simplified version of the *Input Form* pattern. On the left hand side, the interface of the patterns is displayed showing that only one parameter "Number" is expected. This parameter determines the number of elements to be entered.

The middle part of Fig. 5 shows the formalization of this pattern, which consists of a mixture of XUL and velocity template code. The variable $NUMBER is used to determine the number of iterations of the #foreach loop. Within the loop XUL code for displaying the Input fields and labels is produced. Eventually, the outcome of this template (the instantiation of the pattern), which consists of "pure" XUL code can be rendered to a UI fragment of the target platform. The right part of the illustration shows the screenshot of the result of rendering the *Input Form* pattern instance to Windows XP desktop platform.

Please note that, in practice, the *Input Form* presentation pattern is significantly more complex. Since it must embody information and parameters for the types of data input or the internal alignment of the interaction objects.



```
<window>
 <table>
#foreach($i in [1..$NUMBER])
  <tr>
   <td><label value="Input $i"/></td>
   <td><textbox id="input$i"/></td>
  </tr>
#end
 </table>
 <button label="Submit"/>
</window>
```

Figure 5. Different views on the Input Form presentation pattern.

3.3 Patterns and the Layout Model

In the layout model, the various XUL fragments of the presentation model are merged together resulting in aggregated XUL code. The loosely connected XUL pieces are nested and associated together. The way these fragments are merged together depends on the overall "layout" of the entire application.

Usually a Web site or a GUI consists of several pages or windows**.** In order to maintain a consistent feeling across them, the same basic layout or floor plan should be kept throughout the entire interface. Depending on the purpose, the complexity, the in-house style and other attributes, a certain basic layout for the UI is chosen. Selecting a basic layout style usually determines the positioning of navigational elements such as search elements and menus or the size and position of information containers. As a result of this merging process, the style attributes of the UI elements -which were not set- are bound to concrete values. Patterns such as *Column Layout*, *Liquid Layout* or *Card Stack* are used to determine the structure of the layout model.

As Velocity templates can be used to generate XUL fragments (presentation model) they can also be used to aggregate XUL code. Therefore layout patterns are formalised as Velocity XUL templates as well, and the instantiation of these patterns consists of pure XUL code.

Eventually the established layout model (XUL code) is used as input for the automatic generation process in which the concrete interface is generated. Please note that the same layout model can be rendered to different target platforms such as Java Swing and Mozilla /Netscape. XUL has its focus on window-based user interfaces. At the moment XUL specifications cannot be rendered to multiple user interfaces including small devices.

4. RELATED WORK

In [16], van Welie describes how patterns can be used as tools for User Interface Design. He recognised that different kinds of patterns should be formatted in a way, which promotes best its purpose. Within our framework patterns are intended to describe model fragments. Each model is described differently and thus we have introduced different kinds of patterns which have their own formalisation.

According to van Welie's patterns are applicable in different contexts and for each context the adaptation of pattern is necessary. We have adopted this principle and attributed our patterns with variables, which are placeholders for the particular context of use, in which the pattern will be applied.

Van Welie also published a pattern language for interaction design [17]. However all patterns are documented in an informal, narrative way which makes it nearly impossible to implement them in development tools. The overall goal of our approach is the computer aided generation of models, which incorporates patterns as building blocks for re-use. Therefore in this paper we have suggested a possible formalization of patterns.

In [8], Molina also recognised that the mostly-informal description of today's patterns is not suitable for processing them by tools. Within his Just-UI framework, a more precise description of patterns is necessary, which can be interpreted by software tools like validators or code generators. A set of conceptual patterns which realise so-called interaction units is proposed. According to Molina frequent scenarios in user interfaces can be specified with very little effort by combining these interaction units. Molina's patterns are closely connected to the underlying domain object model. They focus on object manipulation and visualization. Compared to our approach Molina's conceptual patterns apply primarily to our presentation model. However, in his work it is also shown how a task description can be extracted and hence the patterns can be applied to underlying task model as well.

Trætteberg [13] recognized that multiple representations should be used in the UI design process and that a uniform modelling language must support the transition between the various presentations. In his work, he suggested RML, TaskMODL and DiaMODL as interlocking fragments of a uniform language for task, domain and dialog modelling. As TaskMODL is quiet similar to our task model DiaMODL is based on the Pisa Interactor and the UML Statecharts notation. Interactors are used to describe the functionality and behaviour of concrete interaction objects, whereas statecharts model the information flow and activation and deactivation of interactors [13].

On the contrary, our dialog model only groups tasks to dialog views and defines transitions between the various dialog views. This allows an earlier generation of a non-functional prototype and thus, earlier user evaluation and earlier iterations. In comparison to Trætteberg's work one could say that within our dialog model the generic interactor is assigned to each task.

Within our framework the definition of interaction objects is described by the presentation model, whereas Trætteberg specifies interactors already during dialog modelling. He uses Statecharts to model the dynamic behaviour including the information flow. At the moment our framework focuses on the generation of non-functional interface prototypes. Thus, the issue of modelling the dynamic behaviour and the information flow between interactors has not been tackled in this work.

In parallel to our approach Trætteberg also suggested to formulate model fragments as patterns in order to facilitate the re-use. In particular he points out the need for patterns in order to describe the mapping between concrete dialog elements to abstract interactors (Presentation and Layout Patterns) and the mapping from tasks to dialogs (Dialog Patterns).

5. CONCLUSION

In this paper, we demonstrated how patterns could be used in conjunction with models to support the UI development process. The core ideas we introduced were highlighted by some examples.

In our model-based framework the application of patterns has a number of advantages. First, they can reduce the time required for UI engineering since for many of the common problems, some pattern solutions already exists. Moreover, a consequent use of UI patterns help in the comprehension of the system for future maintenance.

In particular we have shown which patterns are suitable for several models. In the case of presentation and layout patterns we have suggested a possible formalisation of patterns using Velocity XUL templates. Even though the validity of our approach can not be formally proven, through the realization of the TaskPatternWizard we have experienced that the concept of patterns is applicable and realizable at least for the task model. Furthermore we are progressing in developing a tool that processes and applies dialog patterns.

For the future we aim to develop an all-embracing tool set that supports the integration of patterns into all steps of our model-based framework. We wish to ground our pattern-driven UI engineering methodology as solidly as possible on empirical data and theoretical principles. Furthermore we will validate and compare design patterns with usability tests, particularly for new and experimental patterns and extend our framework to be able to generate functional UI prototypes. In particular we will try to model the information/data flow between the various UI elements.

REFERENCES

[1] Balzert, H., *From OOA to GUIs: The JANUS System*, Journal of Object-Oriented Programming, Vol. 8, No. 9, February 1996, pp. 43-47.

[2] Breedvelt, I., Paternò, F., and Severiins, C., *Reusable Structures in Task Models*, in M.D. Harrison, J.C. Torres (eds.), Proceedings of 4th International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'97 (Granada, 4-6 June 1997), Springer-Verlag, Vienna, 1997, pp. 251-265.

[3] Eisenstein, J., Vanderdonckt, J., and Puerta, A., *Model-Based User-Interface Development Techniques for Mobile Computing*, in J. Lester (ed.), Proceedings of 5th ACM International Conference on Intelligent User Interfaces IUI'2001 (Santa Fe, 14-17 January 2001), ACM Press, New York, 2001, pp. 69-76.

[4] Forbrig, P., Dittmar, A., Reichart, D., and Sinnig, D., *User-Centred Design and Abstract Prototypes*, in Proceedings of BIR'2003 (Berlin, September 2003), SHAKER, 2003, pp. 132-145, accessible at http://www.dsinnig.de/pdfs/User_Centred_Design.pdf

[5] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns: Elements of Object-Oriented Software*, Addison-Wesley, Boston, 1995.

[6] Mori, G., Paternò, F., and Santoro, C., *Tool Support for Designing Nomadic Applications*, in Proceedings of the 8th ACM International Conference on Intelligent User Interfaces IUI'2003 (Miami, 12-15 January 2003), ACM Press, New York, 1993, pp. 141-148, accessible at http://portal.acm.org/citation.cfm?doid=604045.604069

[7] McCall, J., Richards, P., and Walters, G., *Factors in Software Quality*, Three Volumes, NTIS AD, November 1977.

[8] Molina, P., Belenguer, J., and Pastor, O., *Describing Just-UI Concepts Using a Task Notation*, in J. Falcão e Cunha, N.J. Nunes, J. Jorge (eds.), Proceedings of 10th International Workshop on Design, Specification and Verification of Interactive Systems DSV-IS'03

(Madeira, 4-6 June 2003), Springer-Verlag, Berlin, 2003, pp. 218-230.

[9] Paternò, F., *Model-Based Design and Evaluation of Interactive Applications*, Springer-Verlag, Berlin, 2000.

[10] Pressman, R.S., *Software Engineering, A Practitioner Approach*, McGraw-Hill, Berkshire, 2001.

[11] Sinnig, D., Javahery, H., Forbrig, P., and Seffah, A., *The Complicity of Model-Based Approaches and Patterns for UI Engineering*, in Proceedings of BIR'03 (Berlin, September 2003), pp. 120-131, accessible at http://www.dsinnig.de/pdfs/BIR_Pattern_Models.pdf

[12] Souchon, N., Limbourg, Q., and Vanderdonckt, J., *Task Modelling in Multiple Contexts of Use*, in P. Forbrig, Q. Limbourg, B. Urban, J. Vanderdonckt (eds.), Proceedings of 9th International Workshop on Design, Specification and Verification of Interactive Systems DSV-IS 2002 (Rostock, 12-14 June 2002), Lecture Notes in Computer Science, Vol. 2545, Springer-Verlag, Berlin, 2002, pp. 59-73.

[13] Trætteberg, H., *Dialog Modelling With Interactors and UML Statecharts –A Hybrid Approach*, in J. Falcão e Cunha, N.J. Nunes, J. Jorge (eds.), Proceedings of 10th International Workshop on Design, Specification and Verification of Interactive Systems DSV-IS'03 (Madeira, 4-6 June 2003), Springer-Verlag, Berlin, 2003, pp. 346-361.

[14] Trætteberg, H., *Using User Interface Models in Design*, in Ch. Kolski, J. Vanderdonckt (eds.), Proceedings of 4th International Conference on Computer-Aided Design of User Interfaces CADUI'2002 (Valenciennes, 15-17 May 2002), Kluwer Academics Publishers, Dordrecht, 2002, pp. 131-142.

[15] Vanderdonckt, J., Limbourg, Q., and Florins, M., *Deriving the Navigational Structure of a User Interface*, in M. Rauterberg, M. Menozzi, J. Wesson (eds.), Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2003 (Zurich, 1-5 September 2003), IOS Press, Amsterdam, 2003, pp. 455-462.

[16] van Welie, M., van der Veer, G.C., and Eliens, A., *Patterns as Tools for User Interface Design*, in J. Vanderdonckt, Ch. Farenc (eds.), Proceedings of International Workshop on Tools for Working with Guidelines TFWWG'2000 (Biarritz, 7-8 October 2000), Springer-Verlag, London, 2000, pp. 313-324.

[17] van Welie, M., *Patterns in Interaction Design*, 2003, accessible at http://www.welie.com.

[18] XUL, 2003, accessible at http://www.xulplanet.com/