PIM Tool: Support for Pattern-Driven and Model-Based UI Development

Frank Radeke[1,2], Peter Forbrig[1], Ahmed Seffah[2], and Daniel Sinnig[2]

[1] University of Rostock, Software Engineering Group, Institute of Computer Science, Albert-Einstein-Str. 21, 18059 Rostock, Germany
{pforbrig, frank.radeke}@informatik.uni-rostock.de
[2] Concordia University, Human-Centered Software Engineering Group, Department of Computer Science and Engineering, 1455 de Maisonneuve Blvd West, Montreal, QC, Canada, H3G 1M8
seffah@cse.concordia.ca, dsinnig@encs.concordia.ca

Abstract. Model-based approaches describe the process of creating UI models and transforming them to build a concrete UI. Developers specify interactive systems on a more abstract and conceptual level instead of dealing with low level implementation. However, specifying the various models is a complex and time consuming task. Pattern-based approaches encapsulate frequently used solutions in form of building blocks that developers may combine to create a user interface model. Thus they enforce reuse and readability and reduce complexity. In this paper we present a comprehensive framework that unites model-based and pattern-driven approaches. We introduce the "Patterns In Modelling" (PIM) tool, that implements this framework. We will demonstrate the functioning of the tool by using an illustrative example. We primarily focus on the creation of the task model and give a brief outlook how patterns will be applied to the other levels within the framework.

Keywords: User interface design, UI pattern, UI model.

1 Introduction

Software applications are becoming more and more complex. User interfaces are no exception of this rule. Additionally, with the increasing importance of other platforms besides the desktop PC, today's software applications must be developed in a way that they can be easily adapted to platforms like PDAs or Mobile Phones. Developing software on a more abstract and conceptual level can cope with these challenges. This idea is followed by many of the model-based approaches. Nevertheless, specifying the various models and linking them together is still a time consuming and complex task. Specifying the models while using frequently used solutions in form of patterns can reduce complexity and enhance reuse and readability.

Based on these ideas we introduce a pattern driven model-based framework, which allows the application of patterns as building blocks to the different UI models. Afterwards we will introduce the "Patterns In Modeling" (PIM) tool, which aims to support the user with the application of patterns to UI models. We will primarily

K. Coninx, K. Luyten, and K.A. Schneider (Eds.): TAMODIA 2006, LNCS 4385, pp. 82–96, 2007.

focus on the application of task patterns to task models since this step has been already fully implemented in the tool. Additionally we give an outlook on how to apply patterns to the other models of the approach. In doing so, we will use an illustrative example to motivate our approach. Finally we will exhibit future avenues for the development of the framework and the tool.

2 Related Work

There exist a couple of tools that support either model-based UI design or pattern-driven approaches. Figure 1 mentions some of them. The PIM tool, which is introduced in this paper, aims to combine both approaches into one tool and hence to bridge the gap between model-based development and pattern-driven design.

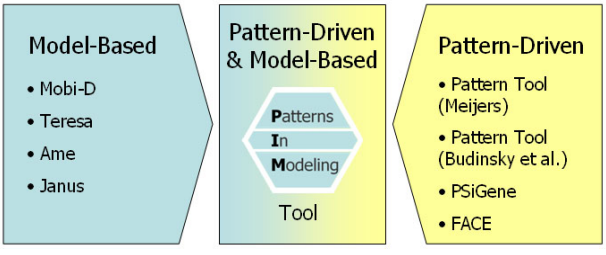

Fig. 1. PIM Tool – Bridging the gap between model-based and pattern-driven approaches

Key idea of model-based approaches is to specify the system on an abstract level using different models. Each model describes a specific aspect of the application.

MOBI-D (Model-Based Interface Designer, [10]) is a software environment for the design and development of user interfaces from declarative interface models. The MOBI-D system supports the user interface developer to design the user interface through specifying task, domain, user, dialog, and presentation models. Internally, the MIMIC modeling language is used to define the various models. MIMIC is the forerunner of XIML and supports the declaration of a so-called design model. This design model describes dynamic relationships between entities of the other declarative models.

TERESA is a semi-automatic environment developed to generate the concrete user interface for a specific type of platform based on the task model. It is composed of a number of steps allowing designers to start with an envisioned overall task model of a multiple user interface application [6] and then derive concrete user interfaces for multiple devices. In particular, TERESA distinguishes four main steps: First a single high level task model of a multi platform application is created. Based on this system task models for specific platforms are defined. The system task model is then used to obtain an abstract user interface, which is composed of a set of abstract interactors. Finally, the platform dependent UI is created by mapping the abstract interactors to platform specific interaction techniques.

The AME and JANUS systems both emphasize the automatic generation of the desired user interface from an extended object-oriented domain model. During this automatic generation process, both systems make use of different comprehensive knowledge bases. They do not include any other declarative models. Both systems generate user interface code for different target systems like C++ source code in order to link it with UI toolkits [11].

In pattern-based approaches frequently occurring solutions in form of patterns are used as building blocks to create the system. In the "Pattern Tool" [7] design patterns are regarded as coarse-grained blocks or fragments for UI design. Accordingly, the resulting system is represented by a hierarchical fragment model. The patterns within this approach are mainly based on the design patterns proposed by Gamma et al. [5] but not restricted to them. Patterns are instantiated by being cloned from a prototype. Sub-fragments of this resulting fragment are then replaced by customized sub-fragments according to the current context of use.

The design patterns of Gamma et al. are also used in the pattern tool introduced by Budinsky et al. [3]. The tool provides specific information for each pattern helping the user to choose the right pattern. The user can enter specific information in order to generate a custom implementation of a pattern, in form of program code.

As the previously described pattern-based tools also the FACE (Framework Adaptive Composition Environment, [8]) tool uses design patterns to develop the application. Schemas are used to show the structure of patterns on an abstract level. The schemas are connected to classes that represent the implementation of a pattern.

PSiGene [12] is a pattern-based component generator for applications in the domain of building simulation. Code templates, which are associated with each pattern, are used to generate the final application code. The user starts with the creation of a class model. Then predefined simulation patterns are assigned to the elements of the model.

3 Integrating Model-Based and Pattern-Driven Design

Design models provide a more abstract and conceptual view on software. Describing software through models on different conceptual levels can reduce complexity and distraction by low level implementation details. In the literature various models like task, domain, user, device, platform, application, presentation, dialog or navigation model are discussed [15]. In this paper we will concentrate on the following models:

- The task model is a hierarchical decomposition of tasks into sub-tasks until an atomic level is reached. The execution order of the task is defined by temporal relationships among peer tasks [15].
- The dialog model describes the human-computer interaction. It specifies when the user can invoke functions, when the user can select or specify inputs and when the computer can query information. Task and dialog model are closely related since tasks are assigned to different dialog states within the dialog model.

- The presentation model describes which UI elements appear in different dialog states of the dialog model. It consists of a hierarchical decomposition of the possible screen display into groups of interaction objects [10]. A relation between dialog and presentation model can be constructed through linking the abstract UI elements with the dialog states in which they appear.
- The layout model finally assigns style attributes like size, font or color to the abstract UI elements of the presentation model [13].

In model-based UI design the user interfaces are specified by creating and transforming UI models and linking them with each other (MBUID, e.g. [14]). The creation of the models and linking them together, however, is a tedious and time-consuming activity. Even for relative simple applications the models quickly become complex. Moreover, it lacks an advanced concept of reuse, besides copy-and-paste-reuse, of already modelled solutions [4]. We believe that patterns, which describe generic solutions to common problems in a specific context [5], can be employed to avoid these disadvantages. Additionally extending the model-based approach with patterns can enhance reuse, flexibility and readability.

Figure 2 integrates the ideas of the former paragraphs in a pattern-driven and model-based approach. The left side shows the task, dialog, presentation and layout mode1, and the mappings between these models. These models are considered as one integrated model, which is collectively referred to as UI multi model. The right side of figure 2 shows the UI patterns, which are employed to establish the UI multi model. A UI pattern contains one or more model fragments as visualized by the hexagons. If a pattern contains only a model fragment of one specific type we refer to this pattern as task pattern (e.g. TP1), dialog pattern (e.g. DP1), presentation pattern (e.g. PP1) or layout pattern (e.g. LP1), depending on the type of model fragment. Patterns that contain two or more model fragments are referred to as multi patterns (e.g. MP4). While multi patterns represent in general less abstract solutions, specific type patterns represent more abstract solutions, which can be employed in different contexts.

Task model fragments are used to describe task fragments, which frequently appear. They are used as building blocks for the gradual development of the task model. Dialog model fragments suggest how tasks should be grouped to dialog views and how transitions between dialog views should be defined. Presentation model fragments provide predefined sets of abstract UI elements in order to create the presentation model. Finally, layout model fragments can be used to assign certain predefined styles to the abstract UI elements in order to establish the layout model. Beside the model fragments the patterns may contain predefined mappings between the different model components.

Additionally it is possible to composite patterns using sub patterns. Thus, more abstract patterns can be used to create more specific UI patterns. Figure 3 visualizes such a pattern-sub-pattern relation. The task model fragment of pattern 1 is established using the task model fragments of pattern 2 and pattern 3. Moreover, pattern 1 uses pattern 3 as sub pattern to establish the dialog model fragment.

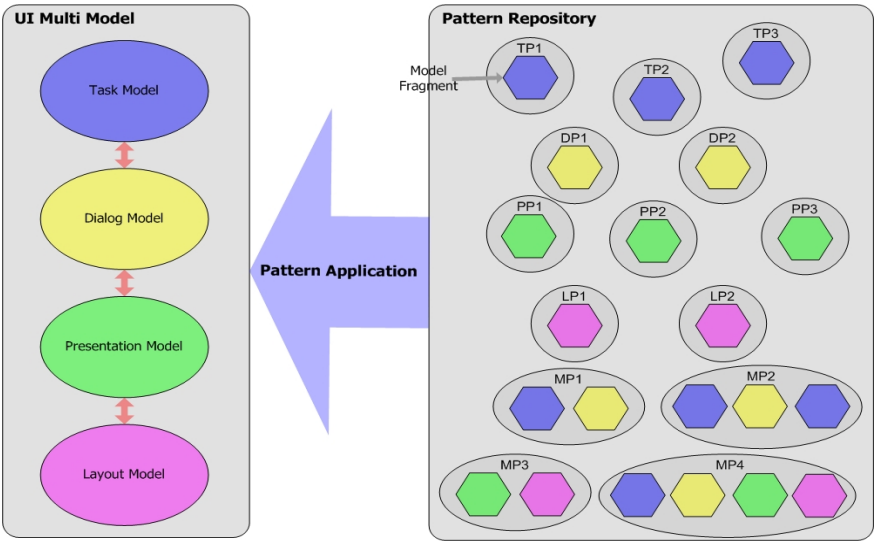

Fig. 2. Pattern-Driven and Model-Based Approach

In order to model the solution stated in the pattern in a generic and reusable fashion, patterns may contain variables and a flexible structure, which have to be adapted to the current context of use. This adaptation takes place during the process of pattern application as visualized by the arrow in figure 2. The pattern application process entails the steps: pattern selection, pattern instantiation and integration of the instantiated pattern into target models.

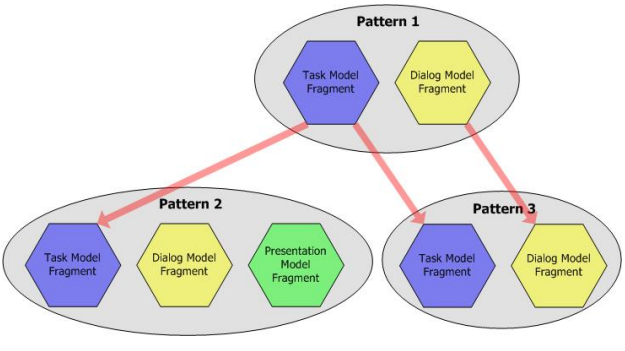

Fig. 3. Pattern-Sub-pattern relations

In the next section, we show how patterns are generally applied to the different UI models. Moreover, we will introduce a tool, which supports the user by applying patterns to the different models of our approach. Primarily we will focus on the task level, which is fully implemented in the tool.

4 PIM Tool: Support for Applying Patterns in Model Construction and Transformation

In the previous section, we introduced a pattern-driven model-based framework, which makes use of patterns to facilitate the creation of the UI multi model. Since patterns are abstract and generic solutions the need to be instantiated. Generally the process of pattern application consists of four steps.

1. *Identification*
   A subset M' of the target model M is identified. This subset will be replaced and extended by the instance of the pattern, which is selected in the next step.
2. *Selection*
   In this step an appropriate pattern is selected to be applied to M'.
3. *Instantiation*
   To apply the pattern to the model it needs to be instantiated. In this step the concrete structure of the pattern is defined and values are assigned to the variable parts of the pattern in order to adapt it to the current situation.
4. *Integration*
   The pattern instance is integrated into the target model. It is connected to the other parts of the model in order to create a seamless peace of design.

Handling these steps depends, to great extent, on the creativity and experience of the designer. At this point efficient tool support can assist the designer in selecting and applying appropriate patterns in order to make the design process more traceable and to increase its effectiveness. For this purpose we introduce the PIM (Patterns In Modeling) tool, which aims to support the user in applying patterns to models by following these steps. Figure 4 shows the user interface of the PIM tool.

The screen mainly consists of three views. (1) The model view, which allows the user to specify the target model (task, dialog, presentation, and layout). In the figure, the task-model label is marked by a bold border frame in order to indicate that the user is working on the task level of the integrated UI multi model. (2) The work view, which is responsible for displaying the selected model. For example, in figure 4 the work view shows the task model of a hotel management application. (3) The pattern view, which shows available patterns for the chosen target model. In this case the pattern view shows a set of task patterns.

In what follows we will use a brief example to illustrate how the tool works. Since currently only pattern application to the task level has been implemented in the tool, we will focus on the task pattern application and mention briefly how pattern application can be realized on the other levels of the approach as well.

4.1 Brief Example: A Desktop Radio Player

To illustrate the functioning of the PIM tool we will now describe a small example, in which we the task model of a simple Desktop Radio Player application is developed.

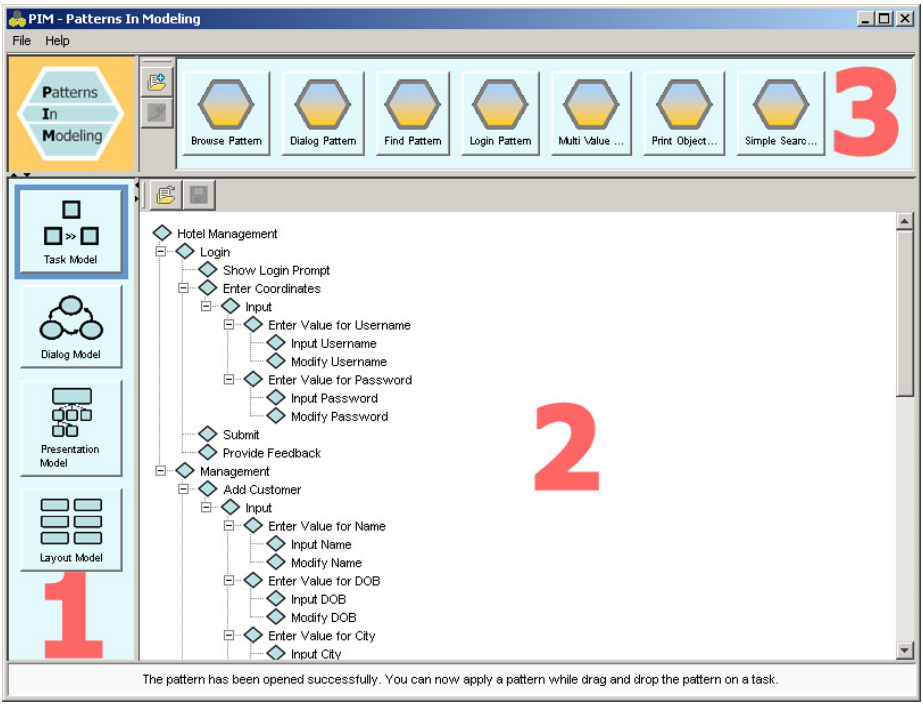

Fig. 4. User Interface of the PIM (Patterns In Modeling) Tool

Table 1 depicts the textual description of the use case capturing the main functionality of the envisioned application.

Following the process of pattern application parts of the task model that can be replaced and extended by task pattern instances are identified. Then appropriate task patterns are selected, instantiated and finally integrated into the original task model. In the following the entire process is described in more detail.

4.1.1 Identification

In order to get started a base model is needed, which can be extended by using pattern instances. Figure 5 shows a very basic task model, visualized in CTTE [9]. The task model entails only the basic task structure of the application. It describes that after logging in the user can search and select a radio station. According to his choice the application will then play music. Instead of refining these tasks now 'by hand' details will be added to the model by applying appropriate patterns. For example, after loading the task model into the PIM tool[1], the "Log In" task may be identified as subset M' of the original model to which a first pattern shall be applied.

[1] Currently the PIM tool accepts only XIML files as model specification. Therefore, in order to reuse the CTT task specification, we used the editor [17] to convert the CTT file to XIML.

Table 1. Brief example: Desktop Radio Player

Many programmers enjoy listening to music while they are working. If you have access to a broadband internet connection online radio stations offer a wide range of music. However, accessing internet, opening the right web pages, log in to the network and force through the web pages of the network in order to choose and listen to a radio station is an extensive business. Thus it would be a nice idea to have a small desktop application that allows logging in to an online radio network and afterwards to search through the radio stations, select a preferred station and that finally allows listening to this station.

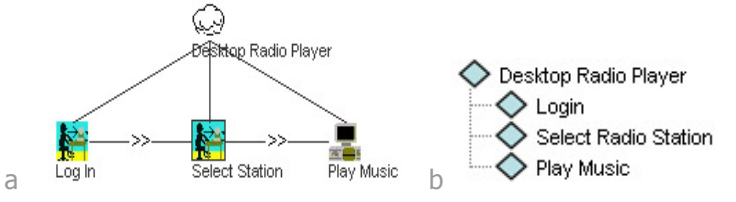

Fig. 5. The task model of the Desktop Radio Player in the CTTE tool (a) and after being loaded into the PIM tool (b)

4.1.2 Selection

Since the task model is selected as working level, the tool offers a set of task patterns that can be applied to the task model, as depicted in the pattern view of Figure 4. With a right mouse click on a task pattern the tool displays more detailed information. For example, table 2 shows the information, which is displayed for the "Log In" pattern.

The format of all patterns follows the uniform format that Alexander [1] proposed for his patterns. Internally the patterns are specified in a XML-based language called TPML (Task Pattern Mark up Language, [13]). As figure 6 shows each pattern specification contains a Name, Problem, Context, Solution, Rational, but also a Body element. Whereas the first five elements are primarily used to help the designer chose an appropriate pattern, the body element specifies the structure of the pattern.

In the PIM tool the "Log In" pattern is dragged and dropped onto the selected "Log In" task of the task model. As a result, the pattern application wizard is launched, which supports the designer performing the next step.

4.1.3 Instantiation

After identifying the "Log In" task as the sub-model M', and after choosing the "Log In" pattern the pattern application process needs to be started in order to create an instance of the chosen pattern. This step is necessary since patterns are generic solutions that have to be adapted to the current context.

The instantiation of a task pattern consists of two steps: (1) The structure of the resulting instance is specified. (2) Values to variable parts of the pattern are assigned.

Table 2. Information displayed for the “Log In” pattern

Problem: The user needs to identify him/herself in order to access secure or protected data and/or to perform authorized operations.
Context: The pattern is applicable if the application manages different types of users fulfilling different roles. The login process is also required if the application supports individual user modeling. An example is the individual customization of the appearance of the front end. In addition, the login feature is necessary if the user wants to create his/her own space, which makes it possible to have users enter that information once and use it again for future visits to the site.
Solution: By displaying a login dialog, the user can identify him/herself. The login dialog should be modal. In addition, the user should login before accessing any personal data or secure features of the application. In order to identify him/herself, the user should enter a combination of different coordinates. After submission, the application provides feedback whether login was successful or not.
Rational: By using the Login Pattern, the user can uniquely identify him/herself. Therefore the applications can provide customized views to the user and authorize the user to perform operations according to the user's role.

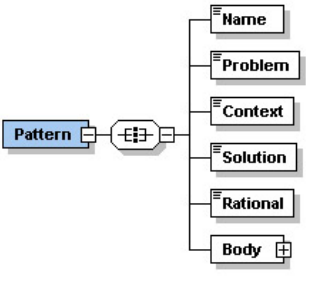

Fig. 6. Pattern structure

The first step includes the selection of optional parts of the pattern. In particular it has to be chosen whether they shall be included or not. Additionally, the number of executions for parts, which can be performed iteratively, has to be defined.

Figure 7 shows the pattern wizard that appears after the “Log In” pattern was dragged and dropped on the “Log In” task. It will guide the user through the pattern application process. The right pane shows the current state of the pattern instance during the whole process. As one can see the “Log In” pattern itself has a simple structure. It describes the “Log In” task as follows: First a login prompt is shown. Then the user enters and submits its coordinates for identification. Finally feedback is provided by the system. Since the “Log In” pattern contains no optional or iterative

parts and also no variables the pattern application process immediately finishes after the user presses the next button in the wizard.

However, as portrayed in the right pane of figure 7 the “Log In” pattern makes use of the “Multi Value Input Form” pattern [2]. The “Multi Value Input Form” pattern is applicable when the user needs to enter a couple of related values. In the context of the “Log In” pattern it is used to enter the values that are necessary to identify the user. Therefore, after completing the process of pattern application for the “Log In” pattern the PIM tool identifies the sub-pattern and prompts the user whether he or she would like to instantiate the “Multi Value Input Form” pattern as well. Figure 8a shows the structure of the “Multi Value Input Form” pattern, which is displayed by the pattern wizard if the user confirms this prompt.

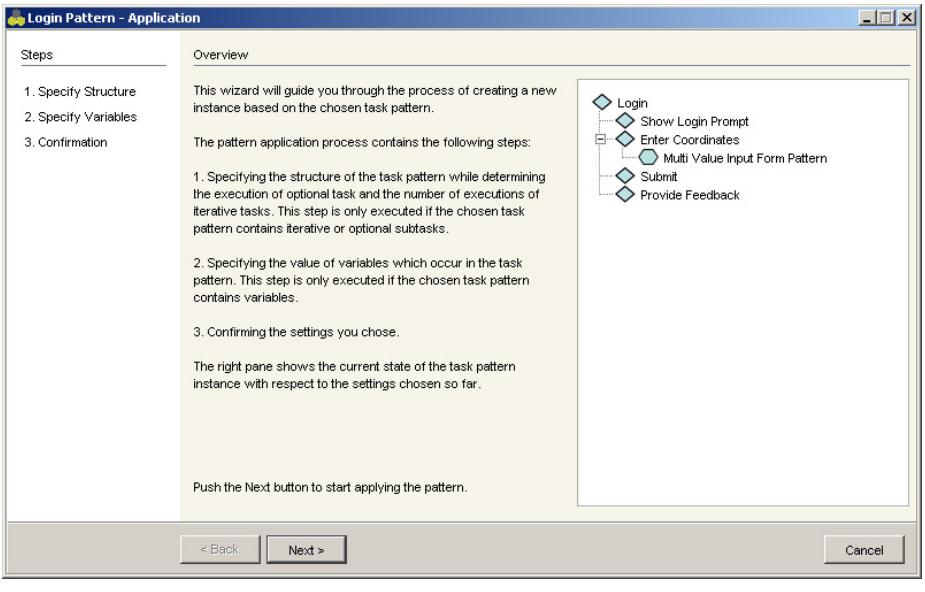

Fig. 7. Wizard for task pattern application to task models

A task icon with a star inside informs the user about the fact that the pattern contains a repetitive task for which the number of executions has to be determined. The number of repetitions depends on the number of values, which are entered within the form. Since it is used within the context of a log-in dialog, the user shall enter two values: ‘username’ and ‘password’. Thus, the structure of the pattern instance in the next screen of the pattern wizard is modified in such a way that the “Enter Value for” task is performed two times. The pattern wizard allows the user to specify the structure of the current pattern instance while directly modifying the pattern instance in the pattern preview. The result is shown in figure 8b. Having a closer look at the obtained pattern instance, it is noticeable that each “Enter Value for” task contains a variable <name> to which a value has to be assigned. Thus, the pattern wizard will

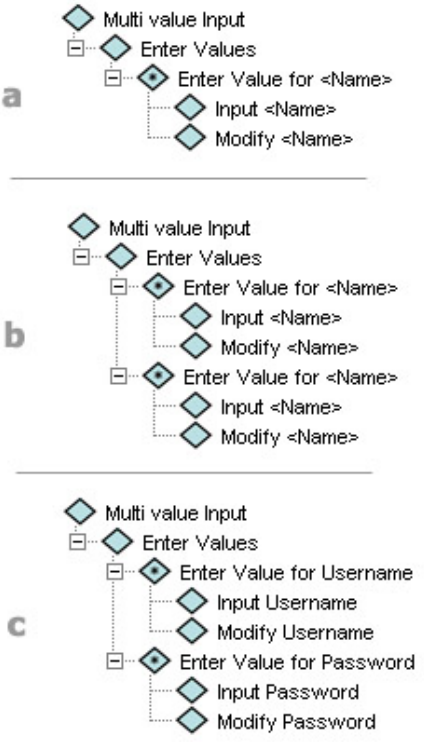

Fig. 8. Instantiation of the "Multi Value Input Form" Pattern

ask the user in the next screen which values should be assigned to these variables. Figure 8c shows the state of the pattern instance, which is displayed by the wizard, after the assignment is done.

In a similar way the "Select Radio Station" task of the original task model is refined by applying an appropriate pattern. In this task the user informs his- or herself about a set of available radio stations and finally chooses one. A suitable pattern for this purpose is the "Browse" task pattern which allows inspecting a set of objects. Figure 9a shows the structure of the "Browse" pattern prior the instantiation process. It contains a variable <object> that describes what kinds of objects are browsed. Since Radio Stations should be browsed the value 'Radio Stations' is assigned to this variable. As a result the pattern instance, which is displayed in figure 9b is received. As portrayed, the "Browse" pattern contains the "Print" pattern as a sub-pattern. It is used to display specific attributes of the browsed objects. Consequently the PIM tool offers a possibility to instantiate this pattern afterwards.

Figure 10a shows the structure of the "Print" pattern. It contains the repetitive task "Print <attribute>" for which the number of repetitions has to be determined. In our example two attributes of the browsed radio stations shall be displayed: The station name and the station genre. Thus it is defined that the "Print <attribute>" task is executed two times. Since the print task contains a variable declaration the value for

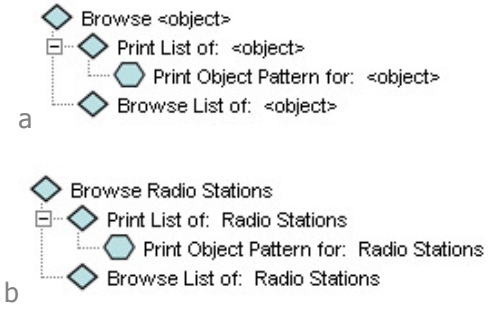

Fig. 9. Instantiation of the "Browse" Pattern

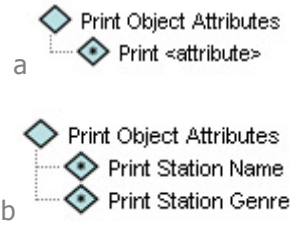

Fig. 10. Instantiation of the "Print" Pattern

the variable <attribute> has to be assigned twice. The instantiation of the "Print" pattern results in the task fragment, displayed in figure 10b.

4.1.4 Integration

Every time a pattern is instantiated as results an instance is received, whose structure and variable parts are adapted to the current context of use. In the last step of the pattern application process this instance has to be integrated into the target model. Therefore, as soon as the pattern wizard is finished, the PIM tool converts the pattern instance to the model format and integrates it into the target model by replacing the part that was selected in the first step.

Figure 11 shows once again the basic task model for the Desktop Radio Player and the resulting task model that was achieved with fewer efforts while making use of pre-defined solutions in form of patterns.

4.2 Outlook

In the future we will further extend the PIM tool so that it will be possible to apply patterns to all models involved in our approach. In the following we will briefly describe applicable patterns for the dialog, presentation and layout model.

A typical dialog pattern that can be applied to the dialog model is the "Recursive Activation" pattern [2]. This pattern is used when the user can activate several instances of a dialog view. It is applicable in many modern user interfaces. An

example is an email application, which allows creating and editing several emails concurrently.

The “Form” pattern is an example of a presentation pattern. It is applicable when the user needs to provide structural logically related information to the application. Thus, it can be used to specify the abstract appearance of the log-in dialog within the Desktop Radio Player application.

Layout patterns can be employed for example for the positioning of the abstract UI elements and for binding concrete values to their style attributes. For the former purpose the “Grid Layout” pattern [16] can be used. This pattern aims to arrange all UI objects in a grid using the minimal number of rows and columns and making the cells as large as possible. An example for the latter purpose is the “House Style” pattern which provides an overall look–and-feel for the entire application.

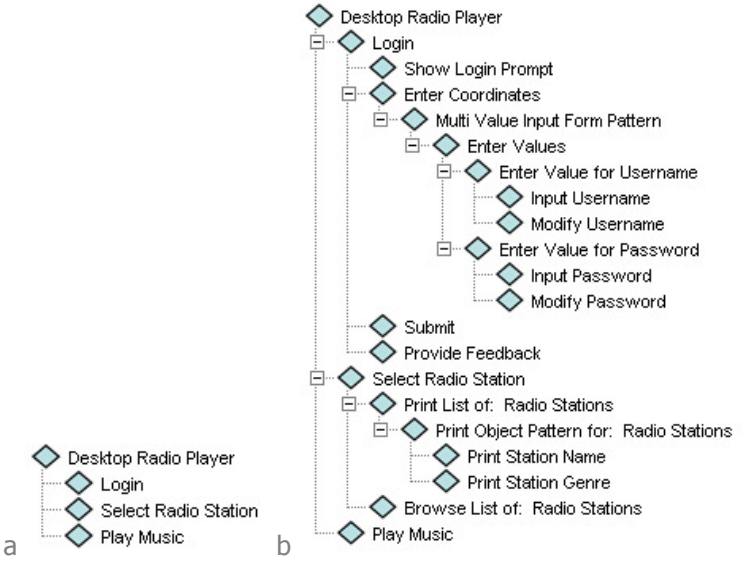

Fig. 11. Original (a) and resulting (b) task model for Desktop Radio Player Example

5 Conclusion

In this paper, we have introduced a novel tool that aims to bridge the gap between pattern-driven and model-based tools. Within the proposed underlying framework instances of patterns are used as building blocks for the creation of UI models and the mappings between them. Employing frequently used solutions in form of patterns reduces complexity and enhances reuse and readability. We demonstrated how the “Patterns In Modeling” (PIM) tool can support software developers in applying patterns to create the UI models. We used an illustrative example to show in detail how the application of task patterns to task models can be supported.

In order to apply patterns on all levels of the UI multi model the proposed framework needs to be formalized. This includes the examination of formal notations for the specification of UI models and the corresponding UI patterns. Presently there exists a multiplicity of UI description languages. Languages like UsiXML and XIML, which support the most common UI models, including the models mentioned in this paper, might be suitable for UI pattern specification as well. This requires extending their schemas to allow the specification of specific pattern features like variables and flexible structures.

References

[1] Alexander, C., S. Christopher, I. Sara, S. Murray, M. Jacobson, I. Fiksdahl-King and S. Angel (1977). *A Pattern Language*. New York, Oxford University Press.

[2] Breedvelt, I., F. Paternò and C. Severiins (1997). *Reusable Structures in Task Models.* In Proceedings of Design, Specification, Verification of Interactive Systems '97, Granada, Springer Verlag.

[3] Budinsky, F., M. Finnie, J. Vlissides and P. Yu (1996). Automatic Code Generation from Design Patterns. *IBM Systems Journal* Vol. 35(No. 2).

[4] Gaffar, A., D. Sinnig, A. Seffah and P. Forbrig (2004). *Modeling patterns for task models.* In Proceedings of ACM International Conference, Prague, Czech Republic.

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

[6] Marucci, L., F. Paternó and C. Santoro (2003). Supporting Interactions with Multiple Platforms Through User and Task Models. *Multiple User Interfaces, Cross-Platform Applications and Context-Aware Interfaces*. London, Wiley: pp. 217-238 London, Wiley.

[7] Meijers, M. (1996). *Tool Support for Object-Oriented Design Patterns*. Master's Thesis in the Department of Computer Science. Utrecht, Netherlands, Utrecht University.

[8] Meijler, D., S. Demeyer and R. Engel (1997). *Making Design Patterns Explicit in FACE, A Framework Adaptive Composition Environment.* In Proceedings of 6th European Software Engineering Conference (ESEC '97), Springer-Verlag.

[9] Mori, G., F. Paternò and C. Santoro (2002). *CTTE: Support for Developing and Analyzing Task Models for Interactive System Design.* In Proceedings of IEEE Transactions on Software Engineering.

[10] Puerta, A. (1997). A Model-Based Interface Development Environment, IEEE Software. 14: pp. 41-47, IEEE Software.

[11] Schlungbaum, E. (1996). *Model-Based User Interface Software Tools - Current State of Declarative Models*. Technical Report 96-30, Graphics, Visualization and Usability Center Georgia Institute of Technology.

[12] Schütze, M., J. P. Riegel and G. Zimmermann (1999). PSiGene - A Pattern-Based Component Generator for Building Simulation. *Theory and Practice of Object Systems (TAPOS)* 5: pp. 83-95.

[13] Sinnig, D. (2004). *The Complicity of Patterns and Model-based UI Development*. Master's Thesis in the Department of Computer Science. Montreal, Canada, Concordia University.

[14] Trætteberg, H. (2001). *Model-based User Interface Design*. PhD Thesis in the Deptartment of Computer and Information Sciences. Trondheim, Norway, University of Science and Technology.

[15] Vanderdonckt, J., E. Furtado, J. Furtado and Q. Limbourh (2003). Multi-Model and Multi-Level Development of User Interfaces. *Multiple User Interfaces, Cross-Platform Applications and Context-Aware Interfaces*. London, Wiley: pp. 193-216 London, Wiley.
[16] Welie, M. (2006). *Patterns in Interaction Design*. http://www.welie.com/ July 2006
[17] Wolff, A., P. Forbrig, A. Dittmar and D. Reichart (2005). *Linking GUI Elements to Tasks - Supporting an Evolutionary Design Process.* In Proceedings of TAMODIA 2005, Gdansk, Poland.