# Patterns in Model-Based Development

**D. Sinnig[1,2], P. Forbrig[1] and A. Seffah[2]**

(1) Software Engineering Group
Department of Computer Science
University of Rostock
18051 Rostock, Germany

pforbrig@informatik.uni-rostock.de

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

**Abstract:** In this paper, we are exploring the roles of patterns in model-based design. In particular, we discuss the fundamental concepts underlying a generic notation for task patterns. Different views of patterns are suggested using UML and Concurrent Task Tree notations. A first classification of patterns according to models is also suggested.

**Keywords:** Task patterns, feature patterns, model-based design, tool support

## 1. Introduction

Historically, user Interface development has been treated as a creative design activity rather than a systematic engineering process. However with the advent of pervasive computing and mobile users the design and the development of User Interfaces has become more and more complex. Thus, user interfaces 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 User Interface development. Within a model based UI development methodology the creation of the task model has been commonly agreed to be a reasonable starting point. We have recently been working on establishing and integrating patterns as buildings blocks for the creation of the task model.

Starting from some general consideration, we will outline different kinds of patterns that can impact the creation of the task model. Then we will introduce a 4-part "strategy" about the process of pattern application followed by an analysis about what a formal notation for pattern for the task model should encapsulate. Finally we will introduce a tool that act as a wizard and guides the patterns user throughout the process of pattern application.

## 2. General Considerations

A pattern can be defined as a reusable *solution* to a recurrent *problem* that occurs in a certain *context* of use. As the re-use of ideas and knowledge becomes more and more crucial, a pattern can be an effective way to transmit experience about recurrent problems in the Software and UI development domain. Therefore the solution should be generic enough to be applicable to many different contexts of use. However in order to illustrate the possible use of a pattern it should encapsulate a very concrete example.

At its best, well and correctly written patterns encapsulate best practices for the Software/UI design process. For software developers unfamiliar with newly emerging platforms, patterns provide a thorough understanding of context of use and examples that show how the pattern applies in different types of applications. Therefore patterns can act as mediators to cross-pollinate software and usability engineering. Moreover pattern catalogs carry a significant amount of reusable design knowledge. However, in order to really effectively re-use the knowledge of patterns tool support is necessary. From this emerges the need for a more formal, machine-readable format for patterns. In the latter we will introduce a format for patterns for the task model that can be interpreted by our tool the "Task-Pattern-Wizard".

Well-document patterns are abstract solution descriptions, which are applicable in different contexts of use. In order to apply patterns they have to be adjusted to the

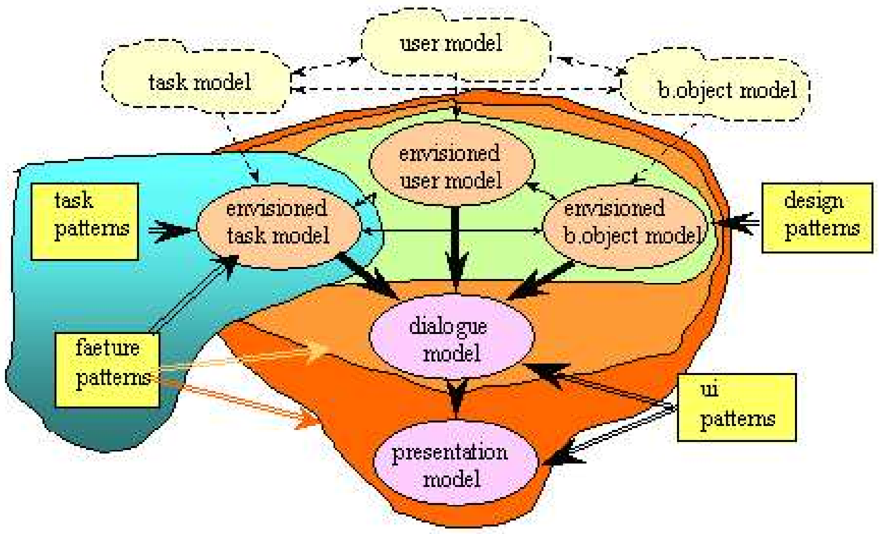

**Figure 1.** *The Complicity of patterns and models for UI development*

current context of use. This adaptation process can be interpreted as a function (A) that takes the pattern (P) and the current context (C) as input and produces a concrete ”sample solution” S.

A (P,C) = S

The application of a sample solution to a task model can be described as a graph transformation, which can be interpreted as a function (F) that takes a task model as a parameter and outputs a modified task model again.

F(T, S) = T*

The task model (T) can be seen as a directed, connected, non-cyclic graph, consisting of set of vertices (V), which represent the tasks, subtasks, and operation. Moreover it consists of a set of edges (E) that represent the relations between the tasks.

T = (V, E)

To be more precise, one can also identify a sub-tree T’ from T and perform a transformation F’(T’,S)=T’* resulting in T*.
Within the next chapter we will be a little bit more specific in specifying these ideas.

## 3. Task Models, Task Patterns and Feature Patterns

### 3.1. Model-Based Approach

In a model based UI design methodology various models are used to describe the relevant aspect of the User Interface. Many facets exist as well as related models: Task, domain (object), user, dialogue and presentation.

*Definitions 1*

A **task model** describes the static and dynamic organization of work.
A **user model** characterizes users and specifies their perception of tasks and organization of work, access rights to data, and their preferences for interaction modalities.
A **business-object model** specifies objects of the problem domain with attributes, methods and relations, as well as the behaviour of these models.
A **dialogue model** describes the structure and behaviour of interaction devices, features, and modalities.
A **presentation model** extends the dialogue model by graphical representations of its elements.

Although it is widely accepted to distinguish between existing and envisioned task models, the same constraint might not hold for the other models. The task, object, and user model have many interrelations. They are not independent dimensions of a system. Each model depends on the other two, as the following examples try to demonstrate:

- A multimedia application cannot be built without media players.
- A multimedia application cannot be built without media players.
- A database is very useful tool for data management but it makes not much sense to use it for the development of a text editor.

*Definitions 2*

**Task** = (Goal, Subtasks, Temp. Relations, Role(s), Artefact, Tool(s), Device)
An **artefact** is an object, which is essential for a task. Without this object the task cannot be performed. The state of this artefact is usually changed in the course of task performance.
A **tool** is an object that supports performing a task. Such a tool can be substituted without changing the intention of a task.
A **goal** is a state of the artefact, which is the intention of performing the task.
A **role** is a stereotype of person who is expected to perform the task.
A **device** describes the stereotype of a computer, which is necessary to perform the task.

Once we assume that the development of interactive systems might be based on envisioned task models, the consequences of this assumption have to be elaborated: Due to the close interrelationship of the models each modification of the task model and an envisioned task model has to be reflected by the other models. Hence, changes to an envisioned task model leads to changes in all the other models. They might become or be envisioned, too. As a consequence, we do not only have to distinguish between an existing and envisioned task model, but also between an existing and envisioned user and object model.

Figure 1 demonstrates how all these models are interrelated and how patterns can support the development process. However, there exist not only relations between models. Each task has relations to other tasks, to objects in the role of artifacts and tools and to users. This is expressed by definitions 2.

We believe that the creation of these models can be driven by the application of patterns. Different types of patterns exist (Design Patterns (Gamma et al., 1995), UI Patterns (Tidwell, 2003), Interaction Design Patterns (Welie, 2003), Process Pattern (Ambler, 1999), Task Pattern (Breedvelt et al., 1997), etc.) Not every kind of patterns is suitable for every model. Figure 1 visualizes which model can be affected by which pattern. For example the use of task- and feature patterns (see the section below) can assist the creation of the envisioned task model. Whereas design patterns (Gamma et al., 1995) are suitable in order to establish the object model. On the other hand UI Patterns are more applicable to less abstract models such as dialogue and presentation.

With respect to the design of interactive systems ”among all these models, the task model has today gained much attention and many acceptances in the scientific community to be the model from which a development should be initiated” (Vanderdonckt and Puerta, 1999). Therefore users - task analysis resulting in a task model is a common starting point in model-based approaches. Possible intentions of the user are captured and activities in order to reach their goals are described. (Paterno, 2000)

From the user task model evolves the envisioned user task model describing from the user's point of view, how activities can be performed in order to reach the user's goal while interacting with the application. (Paterno, 2000) In other words, the envisioned user task model captures the user task and system's behavior with respect to the task-set, in order to achieve a goal. In essence, the system will be viewed through the set of tasks performed by the user, which create input to, and output from the system.

The rest of this paper will focus on the envisioned user task model and on the respectively applicable task- and feature pattern. (See shaded area in Figure 1)

## 3.2. Task- and Feature Patterns

### 3.2.1 Definition

In order to speed-up the creation of a user task model or in order to improve an already existing task model, patterns can 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 ”tool” the user can use in order to fulfill 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 if the feature (stated in the pattern) will adapt. Feature pattern can comprise other sub-feature patterns. Within this paper, we concentrate only on the ”task” - oriented part of feature patterns. However feature always have a visual appearance. Therefore it will remain to our future research to investigate how feature patterns impact the dialog and the presentation model of the user interface. (See also Figure 1)

### 3.2.2 Process of Pattern Application

We have identified the following four sequential steps, in order to practically apply either a task- or a feature pattern to the task model:

- **Identification:** A sub-tree for pattern application within the already existing task tree is identified. (At the moment we restrict ourselves to single nodes as sub-trees.)
- **Selection:** A pattern is selected which is appropriate to be applied.
- **Adaptation:** The pattern will be adjusted to the current context of use resulting in a sample solution. The context can be either captured explicitly through user inputs or implicitly for example through the analysis of an already existing task model.
- **Integration:** The sample solution will be integrated into the current task model resulting in a modified task model. This can affect the task tree in 2 different ways.
    1. Adding a new branch to the tree: This occurs if the pattern introduces a new feature that did not exist before.
    2. Modifying an existing branch: The application of the pattern re-shuffles the particular branch of the task tree in order to improve the arrangement of the user activities with respect to a certain goal, the user has.

### 3.2.3 Notation for Patterns

Taken into consideration that patterns should be flexible to different contexts we suggest that they contain variables. These variables can act as placeholders for the particular context of use. During the process of adaptation these placeholders will be replaced by concrete values representing the particular consequences of the current context of use. As task models are mostly described in hierarchical structures task patterns should follow this principle and describe the task templates in a hierarchical fashion. This also influences the variable concept introduced above, where the scope of a variable will be its definition level and all associated sublevels.

In order to clarify the previously introduced concept of generic patterns containing variable parts, we will now sketch out some examples patterns. We will use different notations like trees for detailed specifications and class diagrams for more abstract considerations. For the sake of simplicity we will only used simplified versions of patterns in this section. Please refer to the next section about how patterns should be developed.

A typical task a user performs in many different applications is to find something. This can range from finding a book at www.amazon.com over finding a used car at www.cars.com until finding a computer in the 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 find Pattern we must abstract from this particulate object and replace it by a variable. Figure 2 gives an impression how the task tree of such a pattern looks like. We will use the notation of CTTE (Paterno, 2000).

Find information can be performed by browsing, searching or using an agent. For more abstract considerations the UML (Booch et al., 1999) notation for parametric classes is used. Figure 3 shows the pattern of Figure 2 in this way. Details of the task structure are omitted. Here the find pattern contains the variable ”Information” which is a placeholder for the particular type of information one is trying to find.

Figure 3 shows, that the “Find” pattern is composed out of the feature patterns “Browse”, “Search” and “Agent”. It is also shown, how the variables of each pattern are interrelated. The value of the variable “Information” of the “Find” pattern will be 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 Figure 4 we have bound the variable “Information” with the value “Book” to create the sample solution “Find Book”; and with the value “Car” to create the sample solution “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 (sub -tasks) has changed

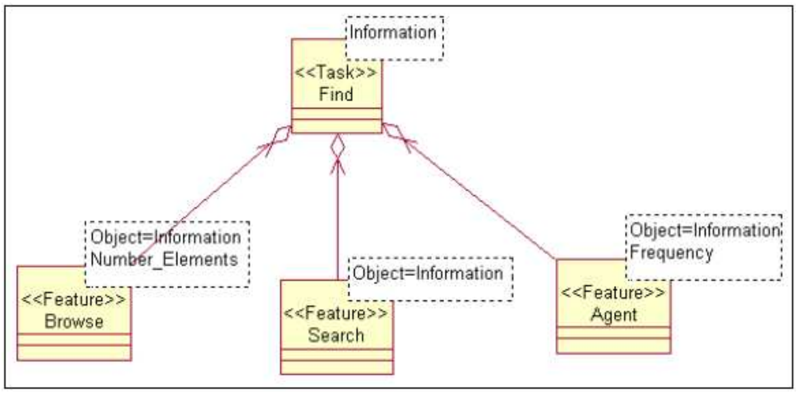

**Figure 3.** *The Find pattern and its sub-patterns*

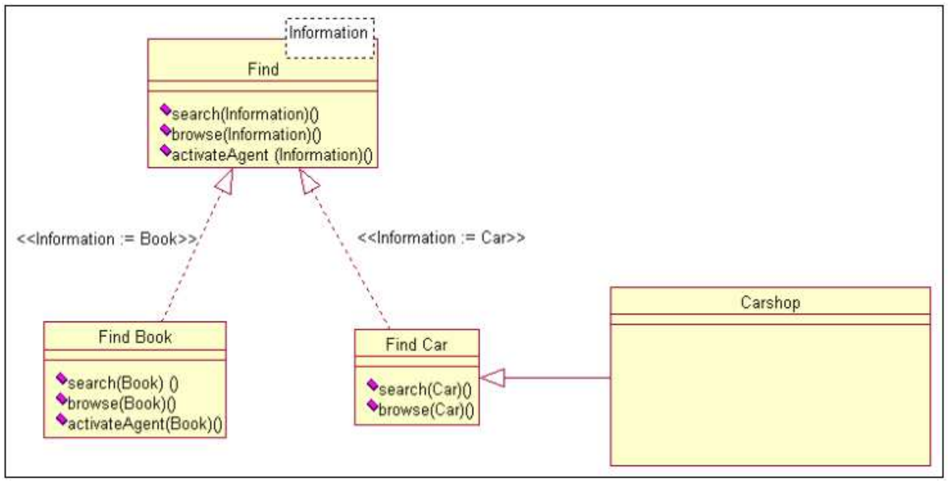

**Figure 4.** *The Find pattern and its sample solutions*

accordingly.

After the pattern adaptation process, the sample solution can be integrated in an already existing task model. In Figure 4 “Find Car” has been integrated into the Car-shop task model. This process of integrations has been visualized using the inheritance relationship and can be interpreted as: The Car-shop has inherited all methods (sub-tasks) from “Find Car”.As mentioned previously, a pattern can be composed out of several sub-patterns.

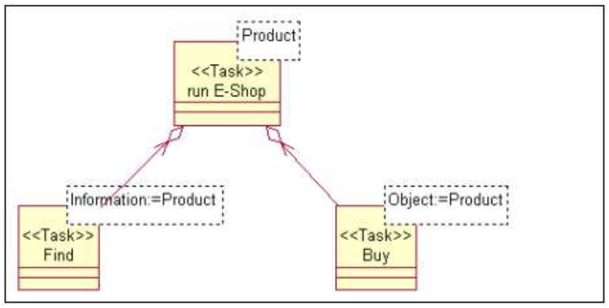

**Figure 5.** *The ”run E-Shop” pattern and its sub-patterns*

In Figure 5 (as well as in Figure 3) we have visualized this pattern - sub-pattern relationship using the concept of class - aggregation. The pattern “run E-shop” consists of the sub-patterns “Find” and “Buy”. If we place patterns in this kind of relationship, we have to pay special attention to the variables of the pattern. A variable, defined at the super - pattern can affect the variables used in the sub-patterns. In Figure 5 the variable “Product” of the “run E-Shop” pattern affects the variables “Information” and “Object” of the sub-patterns “Find” and “Buy”. During the process of pattern adaptation the “Information” and “Object” will be bound with the value of “Product”. We can see that a modification in a high level pattern can affect all sub-patterns.

In order to signal the category or the type of a pattern we use UML class stereotypes. In Figure 5 all patterns are of the type “Task”. Whereas in Figure 4 the “Find” pattern is a task pattern and the patterns “Browse”, “Search” and “Agent” are “Feature” patterns.

In the case of the example of a Car-Shop displayed in Figure 6 the variable Product of the “E-Shop” pattern has been assign with the value “Car” in order to create the sample solution “CarShop”. This value has then been passed to the sub-patterns “Buy” and “Find” in order to assign the variables “Information” and “Object” and subsequently to create the sample solutions “Buy Car” and “Find Car”. In the case of “Find Car” the value of the variable “Information” (“Car”) is used to automatically assign the variable “Object” in all sub-patterns. However in order to fully adapt the sub-patterns “Browse” and “Agent” one will only have to resolve the variables “Number_Elements” and “Frequency”.

The top-down process of pattern adaptation can be greatly assisted by tools, such as the “Task-Pattern-Wizard” (introduced later in this paper). The Wizard runs through the pattern tree and questions the user each time it encounters a variable, which has not been resolved yet. In the case the Car-Shop in Figure 6 the Wizard would question the user for the values of the variables “Product”, “Number_Elements” and “Frequency”. Eventually after resolving all variables the sample solution will be transformed into a concrete task structure. Figure 7 illustrates a cut-out of the modified task model visualized with CTTE. Please note, that the Search task in Figure 7 has been adopted from (Breedvelt et al., 1997).

### 3.2.4 TOOL SUPPORT - THE TASK-PATTERN WIZARD

As mentioned above tool support is necessary in order to apply patterns efficiently. Moreover by integrating the idea of patterns into development tools, patterns can be a driving force throughout the entire UI development process. Therefore we are currently developing a prototype of a task pattern wizard, with which we are aiming to support all phases of the pattern integration, ranging from pattern selection over patter adaptation until pattern integration.

The task pattern wizard is able to read and visualize already existing task - descriptions, that are specified in XIML (XIML, 2003). It is also capable to interpret Task Patterns descriptions that are documented in a prototypical XML based mark-up language. After analyzing the pattern it will guide the user step by step through the pattern adaptation and integration process.

At its best pattern should not only be a vehicle for re-use experiences and knowledge. Beyond this they should also gather user-center design best practices. Ideally a pattern should not be invented at hoc it should furthermore evolve gradually. Therefore we have to find methods in order to validate if a pattern really embodies a ”good” design solution.

In the case of task patterns the natural users behavior should be analyzed in order to create a GOMS (Card et al., 1983) model. GOMS is an acronym for Goal, Operators, Methods and Selection rules. During GOMS analysis tasks are recursively subdivided into sub tasks. Tasks that cannot be split any further are named operators. GOMS comparisons can be used to oppose different design solutions. The execution effort of the operators can be estimated and used to predict the execution time of higher-level tasks, as a total of the individual operator times.

Moreover task simulations can be used to evaluate the appropriateness of patterns. With the help of tools such as the XIML Task Simulator (Dittmar et al., 2003), (Forbrig and Dittmar, 2003) the user can step through possible task scenarios (Pluralistic walkthrough) (Nielsen and Mack,

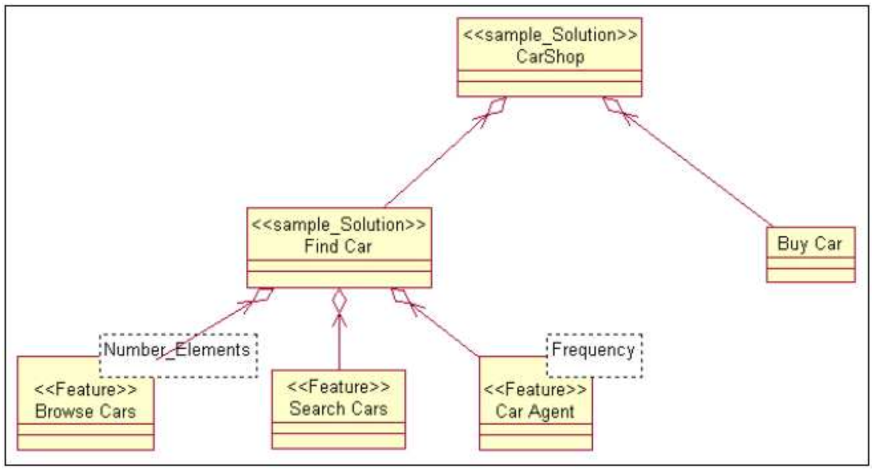

**Figure 6.** *The CarShop sample solution*

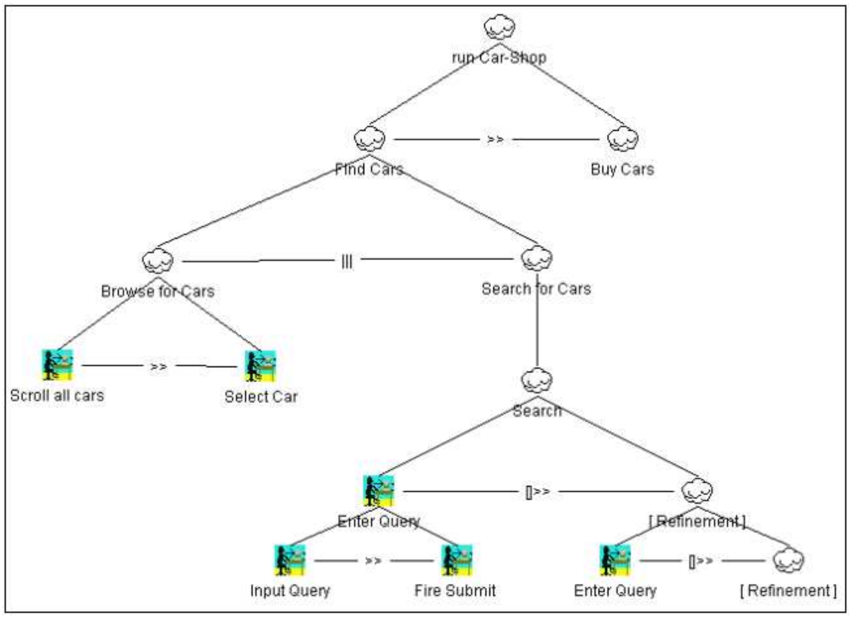

**Figure 7.** *Car-Shop in CTTE*

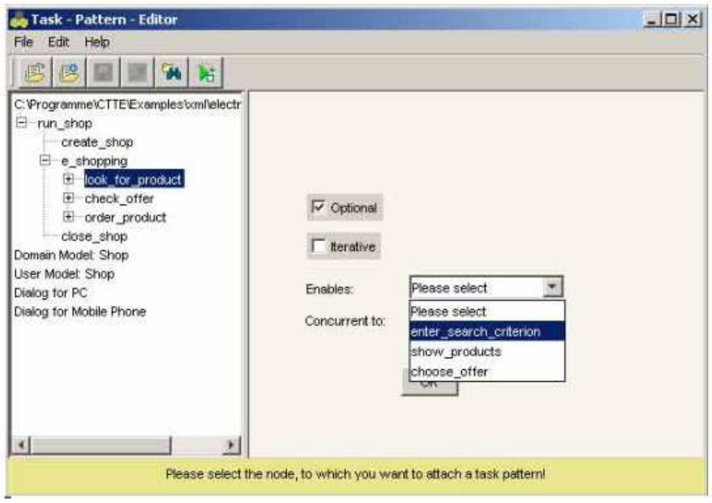

**Figure 8.** *Task-Pattern-Wizard during the Integration phase*

1994) within the scope of the underlying task model that has been established using the patterns to be evaluated. Furthermore tasks can be group to dialogs in order to form a prototypical user interface (i.e. by using the Dialog-Graph-Simulator (Dittmar et al., 2003), (Forbrig and Dittmar, 2003) Using the cognitive walkthrough method (Nielsen and Mack, 1994) 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.

## 4. Conclusions

In this paper, we demonstrated how patterns could be used in conjunction with other models to support the UI development process. Some examples demonstrated the core ideas. During the workshop some more examples will be presented. It can also be discussed which kinds of user interface patterns exists and how they can be integrated in the development process. Our prototype wizard has the potential to cope with such problems as well. One problem is the representation of the user interface. From our point of view XUL is a good candidate for that. UIML and XIML have the problem of lacking tool support at the moment. We expect that the workshop will give us hints in this direction.

## References

Ambler, S. (1999). *Process Patterns: Building Large-Scale Systems Using Object Technology*. Cambridge University Press.

Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *The Unified Modeling Language User Guide*. Addison-Wesley.

Breedvelt, I., Paterno, F., and Severiins, C. (1997). Reusable structures in task models. In *Proceedings of Design, Specification, Verification of Interactive Systems '97*, pages 251 – 265, Granada. Springer Verlag.

Card, S., Moran, T., and Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Assoc.

Chin, D. (1986). User modeling in uc, the unix consultant. In *Proceedings of CHI 1986*, pages 24–28.

Dittmar, A., Forbrig, P., and Reichart, D. (2003). Model-based development of nomadic applications. In *Proceedings of IMC 2003*, Rostock.

Forbrig, P. and Dittmar, A. (2003). Interfacing business object models and user models with action models. In *Proceedings of HCI International 2003*, Greece. Lawrence Erlbaum Associates.

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

Nielsen, J. and Mack, R. (1994). *Usability Inspection Methods*. John Wiley & Sons, New York, NY. ISBN 0-471-01877-5.

Paterno, F. (2000). *Model-Based Design and Evaluation of Interactive Applications*. Springer.

Schlungbaum, E. (1996). Model-based user interface software tools - current state of declarative models. Technical Report 96-30.

Tidwell, J. (2003). Ui patterns and techniques. UI Patterns and Techniques, http://time-tripper.com/uipatterns/index.php.

Vanderdonckt, J. and Puerta, A. (1999). Introduction to computer-aided design of user interfaces. In *Proceedings of the CADUI'99*, Louvain-la-Neuve. Kluwer Academic Publishers.

Welie, M. (2003). Patterns in interaction design. http://www.welie.com.

XIML (2003). http://www.ximl.org.

## A Additional Definitions

*Review: User Task Model*

User Task modeling is an established technique used to analyze, optimize and design human activity and user interfaces. Task models typically describe this data as a hierarchical decomposition of goals, tasks and subtasks and a set of plans that describe the relationships between each set of peer tasks. The nodes of the task tree may contain attributes about the importance, frequency of use and the information needed in order to perform the tasks.

*Review: User Model*

The User model captures the essence of the user's static and dynamic characteristics. It is a widely studied field and we adopt a suitable user model in our research. We can usually model the user knowledge or the user preferences (Chin, 1986). Modeling the user background knowledge is useful for personalizing the format of the information (e.g. using an appropriate language understood by the user). Modeling the user preferences is useful for personalizing the content of the interface (e.g. by filtering the results of a database query, or as an aid to a software agent that proactively notifies the user about interesting information). stated. We can further imagine creating a user model for each type or each individual user or just one user model for the canonical, typical user.

*Review: Object (Domain) Model*

The object model encapsulates the important entities of the particular application domain together with their attributes, methods and relationships (Schlungbaum, 1996). Often the object model is visualized using UML class diagrams.

*Review: 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 functional and logical related elements of the user task model and the domain model (information need to perform the task)

*Review: Presentation Model*

The presentation model expresses the layout and graphic appearance of the user interfaces. It maps the elements of the dialog view to abstract interaction objects, such as menubar, groupbox, listbox), which are device and language independent. The abstract interaction objects are grouped and hierarchical ordered by functionality, only. Moreover attributes are attached to the AIO, such as size, position, color...