Representing User-Interface Patterns in UML

Nuno Jardim Nunes
Universidade da Madeira,
Dep. de Matemática e Engenharias,
9000-390 Funchal, Portugal
njn@uma.pt

Abstract. Software patterns played a major role in object-oriented development, enabling and promoting reuse at higher levels of abstraction. Patterns provided an excellent way to increase the productivity and control in object-oriented information systems’ development. The user-interface is well known to be responsible for a considerable part of the development effort in interactive systems. Yet, we still lack a standard, or at least commonly accepted, notation to express technical representations of UI patterns that convey the solution in an abstract way that can be applied in many different design situations. In this paper we argue that one of the problems preventing the identification and dissemination of UI patterns is the lack of such a modeling notation. We discuss this problem and present an approach based on set of UML extensions specifically adapted for UI design.

1 Introduction

There is a growing interest in the possibility of using patterns [1] in user interface design, development and evaluation [2-6]. Patterns emerged from the ideas of the architect Christopher Alexander and are used to systematize important principles and pragmatics in the construction field. Those ideas have inspired the object-oriented community to collect, define and test a variety of solutions for commonly occurring design problems [7]. Software patterns follow the same principle defined by Alexander, “each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice” [1]. A software pattern is hence, a proven solution for commonly occurring design problem in software.

The idea of applying patterns in Human-Computer Interaction (HCI), and more specifically in User-Interface Design (UID), goes back to the work of Norman [8] and Apple’s Human-Interface Guidelines [9], where patterns are referenced as an influence and inspiration for User-centered Development and UI guidelines. However, only recently, several workshops at UPA’99 [10], INTERACT’99 and CHI’2000 [11] discussed specifically the issue of applying patterns in HCI and UID. Those workshops have confirmed the growing interest, from the HCI community, in using patterns to help leverage the UID process.

If the current research on software patterns enables the dissemination of a widely accepted set of recurring solutions for user-interface design, the impact in software
development (and OO software development in particular) could be considerable. User-interface design is recognized to account for about half of the development effort in interactive systems. Although UI guidelines, and supporting design tools and frameworks, have provided an increased productivity in interactive system development, those tools and techniques leverage fine-grained UI components (for instance GUI widgets and form layout). UI patterns provide a higher-level technique that could contribute to increase the automation and tool support for the interactive aspects of software applications. Moreover, user-interface design is becoming a more complex and demanding task with the advent of multiple information appliances. Therefore, the capability of identifying user-interface patterns, and expressing the solution in an abstract way that is independent of a particular design or implementation is ultimately important.

In the following sections we discuss the problem of conveying UI patterns, in particular the advantages of representing the technical representation (or diagram) underlying UI patterns, through a standard UML language. Section 2 discusses the different interpretations of UI patterns and builds upon a formal proposal to illustrate the main problems faced when representing UI patterns. Section 3 discusses how the Wisdom notation (an extension of the UML for UI design) can be used to illustrate solutions (or diagrams) of UI patterns. Section 4 illustrates this approach with examples drawn from UI a pattern collection publicly available (Wellie’s Amsterdam collection). Finally section 5 presents our main conclusions and future work.

2 UI Pattern Descriptions: Concepts and Definitions

The growing interest in UI patterns generated several concrete collections of patterns that are publicly available [3, 12, 13]. However a consensual pattern language has not yet emerged [6]. There appears to be a lack of consensus over the format and focus for user-interface patterns. In [6] the authors argued that user-interface patterns should focus on the usability aspects that primarily benefit users. The point behind Welie and Troetteberg’s position is that several solutions in user-interface design solve problems that designers (and other stakeholders) have but that don’t necessarily benefit users (for instance a banner in a web page) [6]. This understanding of user-interface patterns is clearly consistent with Alexander’s original definition and the subsequent variation in software patterns [7, 14]. In addition, several authors proposed different formats to represent user-interface patterns. The different formats proposed also follow the initial ideas of Alexander, in particular the description of a UI pattern usually encompasses the following major attributes [1, 5, 7]:

- Identification - including classification and other well-known names for the same pattern;
- Problem - including intent, motivation, applicability and usability problems addressed. The description of the problem usually encompasses concrete applicability examples of the pattern, for instance, scenarios, screenshots, etc.;
- Solution – including a descriptions of the elements that makeup the pattern. The solution doesn’t describe a particular design or implementation because the pattern can be applied in many situations, instead the pattern provides an abstract description of a design problem and how general arrangement of elements solve it;
• Consequences – including results and tradeoffs of applying the pattern and relationship to other patterns (reference to related patterns, variants and sub-patterns).

The attributes for identification, problem and consequences are generally described through natural language or concrete artifacts for the case of depicting the problem. However, the main problem describing patterns is the possibility of depicting the solution in an abstract way that promotes reuse in many analogous, yet different situations. This argument is consistent with the more formal definition of HCI patterns provided by Borchers in [15]. Borchers proposes that a pattern language is a directed acyclic graph, where each node is a pattern and the edges leaving a node are the references. The set of references is the context and each pattern is itself a set consisting of a name, a ranking, an illustration, a problem with forces, examples, the solution and diagram. This syntactical description is augmented with additional semantics describing the concepts of pattern, context, name, illustration, problem, examples, solution and diagram [15].

As we’ve already discussed, the concepts relating to the context, name, illustration and problem are commonly described through natural language. The examples are drawn from concrete real-world solutions that correspond to instantiations of the pattern, usually they correspond to screenshots, scenarios, video or audio snapshots, etc. The main problem is therefore in the concepts related to the solution and diagram. According to Borchers’ formal description, a solution is, semantically, a generalization from the examples that proves a way to balance the forces at hand optimally for the given context [15]. A solution is not simply prescriptive, but generic so it can be applied to different problem situations. According to the same author, the diagram “supports the solution by summarizing its main idea in a graphical way, omitting any unnecessary details”[15]. The main difference between the solution and the diagram in Borchers’ approach is that the former is intended for the users’ audience, while the latter is for a technical audience. This definition is consistent with the results of the various workshops that discussed UI patterns. For instance, in the UI pattern form for the INTERACT’99 workshop, the diagram is called the technical representation and also addresses a technical audience. Furthermore, both approaches to UI pattern descriptions refer the UML as one of the preferable means to represent the diagram (or technical representation).

However, none of the existing pattern collection propose a technical representation for UI patterns. In the next section we build the case for UML technical representations of UI patterns.

3 Technical Representations for UI Patterns with the UML

The possibility of representing the solution underlying patterns through object-oriented notations, such as in [7], was ultimately important for the success of analysis [16] and design patterns [7]. Object-oriented notations, such as the UML, enabled the identification and dissemination of software patterns, because developers have access to a comprehensible notation to depict abstract representations that they could instantiate during implementation.

We argue that one of the problems preventing the identification and dissemination of user-interface patterns is the lack of a modeling notation capable of illustrating, in
an abstract way, the solution that those patterns convey. A closer examination of the existing UI pattern collections reveals that all of them depict the technical representation of the pattern only through concrete examples (for instance screenshots) or through textual descriptions or ad-hoc sketches [3, 6, 12, 13]. Moreover, those descriptions of patterns usually focus on the presentation aspects and neglect the structure of the task underlying the pattern. One notable example, in the other extreme, is provided in [5] where the solution for task patterns is represented using the *ConcurTaskTrees (CTT)* notation, thus being both abstract and concentrating in the task structure. However, Paterno’s approach excludes the presentational aspects of the pattern, which are ultimately important for a complete understanding of the different ways where a UI pattern might be useful.

There are many other notations in the HCI field, in particular those underlying many model-based approaches (for instance [5, 17-19]), that could be used to represent UI patterns. Nevertheless we argue that, on the one hand, those notations are not easily accessible outside the HCI community, in particular to the traditional software developer. On the other hand, those notations are usually highly associated to an underlying UID method or model-based development environment — in a situation very similar to the “method war” that preceded the UML standardization — thus, it would be almost impossible to reach a consensus over one particular notation to express the technical representation of UI patterns. There are other notations that could be candidates for UI pattern representation, in particular those related to web enabled markup languages that became very popular in the last years (for instance HTML forms, DHTML, or the more UI specific AUIML [20], UIML [21] and Xforms [22]). However, hypertext markup is not adequate for modeling purposes. The complex syntax, and the absence of diagrammatic representations, compromise the power and flexibility of markup as an effective modeling approach. Markup languages leverage the semantic aspects in detriment of flexibility and manipulability (by humans) of their notation.

Here we illustrate how technical representations of UI patterns could be expressed through a set of extensions of the UML specifically adapted to UI design, the Wisdom notation [23]. The Wisdom notation adapts several recent contributions in the field of OO&HCI [24] to the UML style and standard, and also proposes new notations to support effective and efficient user-centered development and user-interface design. The Wisdom notation enables the description of presentation aspects of user-interfaces, including support for modeling abstract user-interface components, their contents, containment relationships and navigation. In addition, it supports user-interface behavior modeling with an adaptation of the well-known CTT notation.

The Wisdom notation emerged from previous work on the Wisdom method (a lightweight software development method specifically tailored to develop interactive system by small software companies). Although the notation constructs were developed some years ago, we have plans to redesign the extensions taking into account the forthcoming UML 2.0 standard. Some of the problems with the Wisdom notation (described in [23]) could be solved with the increased support for UML extensions expected to integrate the new release of the standard. However, we believe that notational extensions should be based on actual usage requirements, and applying the Wisdom notation to express UI patterns in one such effort that could highlight the limitations of the existing proposal.
A detailed description of the Wisdom notation is out of the scope of this paper. In the following paragraphs we briefly summarize the extensions required to understand the examples provided in section 4.

To support the Wisdom dialogue model (used to model the dialogue between the user and the software system) the following UML extensions are proposed:

- **<<Task>>** is a class stereotype used to model the structure of the dialogue between the user and the system in terms of meaningful and complete sets of actions required to achieve a goal;
- **<<Refine task>>** is an association stereotype between two tasks denoting that the target class (subtask) specifies the source task (parent task) at a different (lower) level of detail;
- **<<infopass>>** is a dependency stereotype denoting the independent task sends information to the dependent task;
- **<<seq>>** is a dependency stereotype denoting that the dependent task is activated once the first task terminates;
- **<<seqi>>** is a dependency stereotype denoting that the independent task activates the dependent task with information exchange;
- **<<deact>>** is a dependency stereotype that denoting that the dependent task is definitely deactivated once the dependent task terminates.

To support the Wisdom presentation model (used to model the presentation aspects of the user interface) the following UML extensions are proposed:

- **<<Interaction space>>** class stereotype representing the space within the user interface of a system where the user interacts with all the functions, containers, and information needed for carrying out some particular task or set of interrelated tasks;
- **<<navigate>>** is an association stereotype between two interaction classes denoting a user moving from one interaction space to another;
- **<<contains>>** is an association stereotype between two interaction space classes denoting that the source class (container) contains the target class (content);
- **<<input element>>** is an attribute stereotype denoting information received from the user, i.e., information the user can manipulate;
- **<<output element>>** is an attribute stereotype denoting information presented to the user, i.e., information the user can perceive but not manipulate;
- **<<action>>** is an operation stereotype denoting something a user can do in the physical user interface that causes a significant change in the internal state of the system.

For a complete reference to the UML notational extensions used in this paper refer to [23].

As the examples in the next section illustrate, the Wisdom notation provides notational constructs for depicting both the task and presentation aspects of UI patterns. Since the Wisdom notation is based on CTT, there is a clear mapping to the task patterns provided in [Paternò, 2000]. Furthermore, the Wisdom presentation model, notably interaction spaces, provide a means to convey abstract presentation aspects of conventional user-interface patterns. Thus, it is possible to represent in an abstract way a set of already identified patterns, such as the ones provided in the previously mentioned UI pattern collections.
4 Examples of UI Patterns Represented in the UML

Figure 1 exemplifies how the Wisdom notation for the dialogue and presentation models can be used to illustrate the Wizard pattern from the Amsterdam collection [6]. This pattern solves the problem of user wanting to achieve a single goal, which involves several decisions that are not completely known to the user. This particular pattern is widely used in many applications (for instance Microsoft Office applications) to support infrequent tasks that involve several subtasks where decisions need to be made. The Wisdom dialogue model to the left-hand side of Figure 1 defines a minimal sequence of tasks, with the corresponding temporal relationships, that support an abstract UI Wizard. A UI Wizard involves a set of iterative step tasks that can be deactivated (cancel task) at any point in time (that is while performing any wizard iterative step). The iterative step task involves performing a sequence of steps (perform step task) that deactivate the previous step (thus the <<deact>> dependency stereotype between previous and actual step) and also depend on information provided by the previous steps (accordingly the sequence with information passing <<seqi>> dependency stereotype between actual and next step).
To the right-hand side of the figure is a Wisdom presentation model that illustrates how an abstract Wizard can be modeled through two interaction spaces, one for the wizard body and another one for each wizard step (multiple steps are denoted by the $1..*$ cardinality in the <<contains>> association stereotype). Abstract actions (denoted by the <<action>> operation stereotype) are associated with each interaction space denoting typical actions performed in a Wizard pattern (for instance next, back, cancel and finish). As we can see from the example in Figure 1, both the dialogue and presentation models illustrate the pattern without committing to a particular design, implementation technology, platform or interaction technique. For instance, such a model could be used to instantiate a particular Wizard in a web application (a set of DHTML pages), a palmtop (a set of cards) or the obvious desktop wizard in any platform providing a conventional GUI.

Figure 2 exemplifies how the Wizard pattern can be instantiated for a specific and well-known example – the add printer Wizard for Microsoft Windows XP. As we can see from the example the instantiation of the UI Wizard is straightforward. The Wizard body interaction space maintains the common actions for the Wizard patterns and, for each Wizard step, specific interaction spaces are introduced with their corresponding output elements, input elements and actions. A similar approach could be applied to the dialogue model, refer to [5] for similar examples.

Figure 1 provides an example of the Wisdom notations applied to a typical GUI UI pattern; however our approach can also be applied to web-based UI patterns. Such an
example, for e-commerce, is provided in Figure 2 for the Shopping Cart pattern (also available from the Amsterdam collection [6]). The shopping cart pattern solves the problem of users wanting to buy a product; the solution involves providing a shopping cart where users can put their products before they actually purchase them.

The Wisdom dialogue model to the left-hand side of Figure 3 defines the minimal sequence of tasks, with the corresponding temporal relationships, that support an abstract shopping cart. The shopping cart pattern involves shopping for products (denoted by the concurrent independent task continue shopping) and providing details about the current selection of shopping items. At any given point in time the user can change or select the shipping method and change the details of a shopping item (both are mandatory for calculating the total amount due - thus the <<seqi>> temporal relationship). The calculate total task is an example of another possibility provided by the CTT formalism (and accordingly in the Wisdom UML adaptation of CTT), which is task allocation in UI design. Task allocation is denoted by UML tagged values associated to task stereotypes (here omitted in interactive tasks but highlighted in two application tasks by means of the {application task} UML tagged value). In current example, two application tasks are modeled because of their importance supporting the user’s interaction. The calculate total and recalculate subtotal tasks are particularly important because they involve invoking application functionality that might impact the user experience - for instance if both application tasks execute in more than a few seconds the user can perceive an abnormal behavior and give up shopping. This technique provides a solution for modeling UI non-functional requirements that are an important aspect in software development.

To the right-hand side of Figure 3 is a Wisdom presentation model that illustrates the abstract presentation aspects of the shopping cart wizard. This particular UI
pattern can be modeled through two interaction spaces, one for the shopping cart that models information required for the user to enter (denoted by the <<input element>> attribute stereotype – here shipping method) and also important information that supports the user interaction (denoted by the <<output element>> attribute stereotype – here subtotal, total and shipping). As explained in the previous example the <<action>> stereotype enables modeling of abstract actions that the user can perform in each interaction space. Finally, the <<contains>> association stereotype connects the shopping cart interaction space with the contained shopping item interaction space (with a cardinality of 1 or more). The shopping item interaction space also includes a set of input and output elements and actions that enable the user to view (price, description, availability), change (quantity) and act upon (remove) specific shopping items.

**Conclusion**

User-interface patterns are a very important step towards promoting consistency in user-interfaces. User-interface design is becoming a more complex and demanding task with the advent of multiple information appliances. Therefore, the capability of identifying user-interface patterns, and expressing the solution in an abstract way that is independent of a particular design or implementation is ultimately important. Only that way we can promote a language of patterns that supersedes the restrictions of a particular platform. The notational extensions provided in the Wisdom notation for both task and presentation aspects of user-interfaces enable the abstract description of user-interface patterns. Moreover, since the Wisdom notation complies with the UML standard, we can take advantage of enhanced communication with software developers, while also taking advantage of tool support. Access to UI patterns in modeling tools can increase the efficiency of the design process by enabling designers to incorporate different patterns when they are producing the models.

We are currently applying our approach to as many UI patterns as possible to detect possible problems with the expressiveness of the UML extensions proposed as an effective means to depict technical representations. Some of the problems already identified are related to the fact that many patterns (for instance many of the Web patterns in www.welie.com) capture concrete presentation aspects such as position and the use of color. Our UML based approach is clearly not suitable for such class of UI patterns, since the positioning or coloring of modeling elements in the UML is not semantically defined.

As a result of applying the Wisdom notation to a wide set of UI patterns, we aim to raise notational requirements for the next generation of the Wisdom UML extensions that take advantage of enhanced extensibility mechanism provided by UML 2.0. The next release of the UML standard is expected to be a step towards a language of languages, a requirement that ultimately benefits specific modeling requirements such as the ones presented in this paper.
References