CanonSketch: a User-Centered Tool for Canonical Abstract Prototyping

Pedro F. Campos and Nuno J. Nunes

Department of Mathematics and Engineering, University of Madeira
Campus da Penteada, 9000-390 Funchal, Portugal
{pcampos,njn}@uma.pt

Abstract. In this paper, we argue that current user interface modeling tools are developed using a formalism-centric approach that does not support the needs of modern software development. In order to solve this problem we need both usable and expressive notations and tools that enable the creation of user-interface specifications that leverage the design and thought process. In this paper we present the CanonSketch tool. CanonSketch supports a new UI specification language – Canonical Abstract Prototypes (CAP) – that bridges the gap between envisioned user behavior and the concrete user interface. The tool also supports two additional and synchronized views of the UI: the Wisdom UML presentation extension and concrete HTML user interfaces. In this way the tool seamlessly supports designers while switching from high level abstract views of the UI and low-level concrete realizations.

1 Introduction

Model-based user interface design (MB-UID) has been the target of much research during the last decade. However, and despite the success obtained by user interface development tools, approaches based on models are not reaching the industrial maturity augured in the 90's [4].

In a paper presented at a recent Workshop on MB-UID [9], we argued that in order to achieve a stronger market acceptance of modeling tools, a new generation of user-centric tools would have to emerge. The existing tools are focused on the formalisms required to automatically generate the concrete user-interfaces. This legacy of formalism-centric approaches prevents the current tools from adequately supporting the thought and design tasks that developers have to accomplish in order to create usable and effective user-interfaces. Model based approaches concentrate on high-level specifications of the user-interface, thus designers loose control over the lower level details. These problems with MB-UI tools are described in [4]. In particular, those tools suffered from trying to solve the "whole problem" and thus providing a "high threshold/low ceiling" result. The threshold is related to the difficulty of learning a new system and the ceiling is related with how much can be done using the system. Thus, those tools don't concentrate on a specific part of the UI design process and are difficult to learn, while not providing significant results.

In order to overcome these limitations, designers directly use a user-interface builder (a low threshold/low ceiling tool) that provides them with adequate and flexible support for designing the user-interface. Designers that recognize the value of modeling at higher levels of abstraction are forced to use different tools and notations to capture the user-interface specifics at different levels of abstraction – what could be considered as using many low-threshold/low ceiling tools.

Some of the requirements for such tools were also discussed in a recent workshop about usability of model-based tools [11]. Among other issues, the participants at the workshop highlighted the following requirements as paramount to promote usability in tools: traceability (switching back and forth between models, knowing which parts can be affected by changes), support for partial designs, knowledge management (for instance, a class that is selected or modified often is probably more important than classes not often changed) and smooth progression from abstract to concrete models.

In this paper we present a new tool, under development, that tries to leverage the users' previous experience with popular Interface Builder (IB) tools in order to achieve better adoption levels. Our aim is to build a developer-centric modeling tool that applies successful concepts from the most usable and accepted software tools. Instead of defining a complex semantic model and formalisms to automatically generate the user interface (UI), we start by using a simple sketch application and extending it to accommodate the required concepts and tools. The tool supports the creation and editing of Canonical Abstract Prototypes [2] and Wisdom Presentation Models [7]. It is capable of automatically generating HTML interfaces from the Canonical specification. In this initial phase, we are focusing on specifying GUI's for Web-based applications, although conceptually the tool is not restricted to this type of interface, since the languages are platform and implementation independent. However, this allows us to test the main concepts of the tool/language by focusing on a well-known interface type.

This paper is organized as follows: Section 2 relates our work to some approaches for UI design and Section 3 briefly describes the main notation our tool supports: Canonical Abstract Prototypes. Section 4 presents CanonSketch, detailing some of its user-centered features. Section 5 proposes an initial extension to the Wisdom presentation model in order to support the Canonical notation. Section 6 investigates the capability of both notations to express UI design patterns in an abstract way. Finally, Section 7 draws some conclusions on our present work and presents possible future paths to follow.

2 Prototyping and Sketching Interfaces

Rapid prototyping of interactive systems is a technique used in order to assess design ideas at an early stage of the development process. It attempts to foster the collaboration between all the stakeholders involved in the project (managers, end-users, graphic designers, coders...) and to facilitate iterative cycles of reviewing and testing.

Being a de facto standard in the development community, the UML provides a good medium to specify UIs enabling higher acceptance rates and promoting artifact inter-

change between modeling tools. UML class stereotypes have become a very popular alternative to structure the presentation elements of interactive systems [7]. In particular, the Wisdom notation complies with the UML standard, thus enhances communication with software developers. Another strategy, used by the DiaMODL approach, combines this with a strong linkage to concrete UI elements [10]. Other approaches are used in different areas: Hypermedia applications, such as in [13] and [14] and Cooperative System modeling [15].

Prototyping interfaces with electronic sketching tools has also proven successful in systems such as SILK [3] or DENIM [5]. Sketching is believed to be important during the early stages of prototyping, because it helps the designers' creative process: the ambiguity of sketches with uncertain types or sizes encourages the exploration of new designs without getting lost in the details, thus forcing designers to focus on important issues at this stage, such as the overall structure and flow of the interaction [3].

However, widget recognition is hard for these systems [3], since any widget recognition algorithm might be too error-prone. Also, usability tests reported that some users had trouble manipulating and entering text, and understanding how to select, group and move objects.

Calgary et al. [16] describe a framework that serves as a reference for classifying user interfaces supporting multiple targets, or multiple contexts of use in the field of context-aware computing. This framework structures the development life cycle into four levels of abstraction: task and concepts, abstract user interface, concrete user interface and final user interface [16]. These levels are structured with a relationship of reification going from an abstract level to a concrete one and a relationship of abstraction going from a concrete level to an abstract one. As we will see in this paper, maintaining a connection between these levels is well supported in CanonSketch.

Canonical Abstract Prototypes [2] were developed by Constantine and colleagues, after a growing awareness among designers regarding the conceptual gap between task models and realistic prototypes. They provide a common vocabulary for expressing visual and interaction designs without concern for details of behavior and appearance. Moreover, they fill an important gap between existing higher-level techniques, such as UML-based interaction spaces and lower-level techniques, such as concrete prototypes. This is why we chose this notation as our starting point for our modeling tool. In the following section, we briefly describe the Canonical notation.

3 Canonical Abstract Prototypes

Constantine [2] proposes a stable collection of abstract components, each specifying an interactive function, such as inputting data or displaying a notification. Following on the successful path of interface builders, these components can be selected from a palette in order to build abstract prototypes, thus fostering flexibility and modeling usability. Having a standardized set of abstract components also eases the comparison of alternative designs and enhances communication between members of the development team [2].

The symbolic notation underlying Canonical Abstract Prototypes is built from two generic, extensible[1] universal symbols or glyphs: a generic *material* or *container*, represented by a square box and a generic *tool* or *action*, represented by an arrow. Materials represent content, information, data or other UI objects manipulated or presented to the user during the course of a task. Tools represent operators, mechanisms or controls that can be used to manipulate or transform materials [2]. By combining these two classes of components, one can generate a third class of generic components, called a *hybrid* or *active material*, which represents any component with characteristics of both composing elements, such as a text entry box (a UI element presenting information that can also be edited or entered). Figure 1 shows the three basic symbols of the Canonical Abstract notation. For a more detailed look of the notation, please refer to Figure 6.

Fig. 1. The three basic symbols underlying the symbolic notation of Canonical Abstract Prototypes (from left to right): a generic abstract tool, a generic abstract material and a generic abstract hybrid, or active material (taken from [2]).

Although Canonical Abstract Prototypes lack a precise formalism and semantics required to provide tool support and automatic generation of UI, we found the notation expressive enough to generate concrete user interfaces from abstract prototypes. In the following section, we present our tool, including a proof of feasibility in which we generate HTML pages from sketches of Canonical Abstract Prototypes.

4 CanonSketch: The tool

Different tools (business presentation applications and even sticky notes or whiteboards) can be used for creating Canonical Abstract Prototypes. However, in order to assess and benefit from all of the advantages of this notation, software tool support is ultimately needed [2].

CanonSketch aims at providing a usable and practical tool to support Canonical Abstract Prototypes. Starting with an easy to learn notation, developed from real world projects, we built a tool that provides the user a palette of abstract components that can be drawn, grouped, resized and labeled within a drawing space representing an interaction space. The tool supports all the successful features one expects to find in software nowadays, like multiple undo/redo, grid layout, tool tips or send to back/bring to front.

Our tool already supports the creation (at the syntactic level only) of Wisdom interaction spaces [6]. Our aim is to leverage developer experience of the Unified Modeling Language (UML) by designing an extension to the UML that fully supports Canonical Abstract Prototypes. Figure 2 shows a CanonSketch screenshot of the Wisdom view, where the designer is creating a Wisdom presentation model as if she were

[1] Meaning all other components can be derived, or specialized, from these classes.

sketching in a simple drawing application. Figure 3 shows a screenshot of the Canonical view: we can see that there are several palettes of tools available (e.g. for controlling font-size, coloring and grid layout) and an inspector as well as an optional ruler.

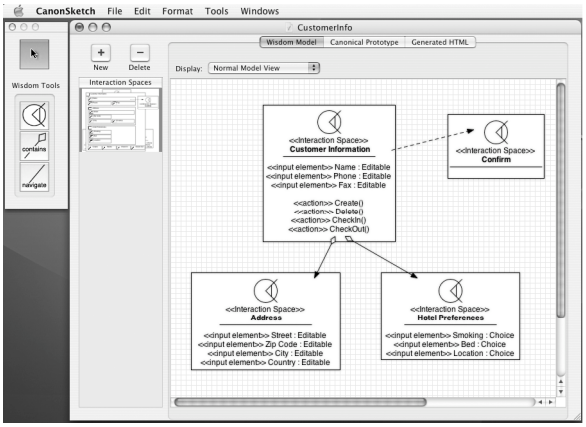

Fig. 2. CanonSketch screenshot: creating Wisdom UML presentation models.

In our path to building a usable modeling tool for UI design, we began with a different approach from the conventional way these tools are envisioned: instead of focusing on the formalisms and semantics, we began with a simple drawing application and built a modeling tool that relies on interaction idioms more closely related to Office applications, as we discuss in the following sections. Our remit here is that we intend to focus on achieving a modeling tool that is as easy to use as a drawing application.

4.1 User-Centered Features

UI tools represent an important segment of the tool market, accounting for 100 million US Dollars per-year [4]. However, there has been a gross decline on the modeling tools market revenue, according to reliable sources such as the International Data Corporation. The lack of usability present in modeling tools is believed to be responsible for this weak adoption [11].

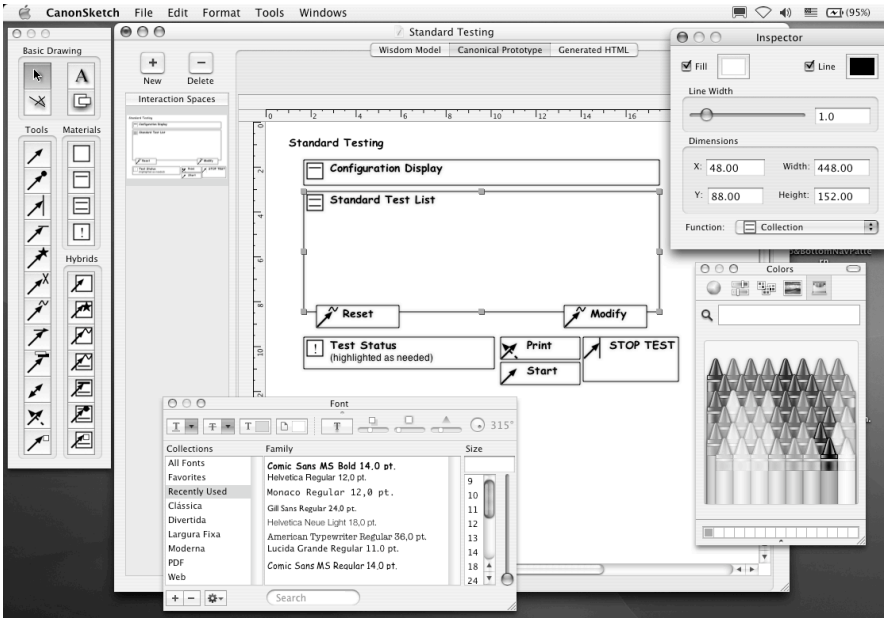

Fig. 3. CanonSketch screenshot: creating and editing Canonical Abstract Prototypes.

A more developer-centered approach was followed in CanonSketch: Figure 4 shows some of the aspects we took into account. Canonical Abstract Prototypes are organized in terms of sequences of interaction spaces that appear as thumbnails of their corresponding specifications. By using this pattern, very common on business presentation applications, we aim at leveraging the existing user experience while also promoting communication and collaboration between developers and clients (who are often businessmen familiar with this pattern).

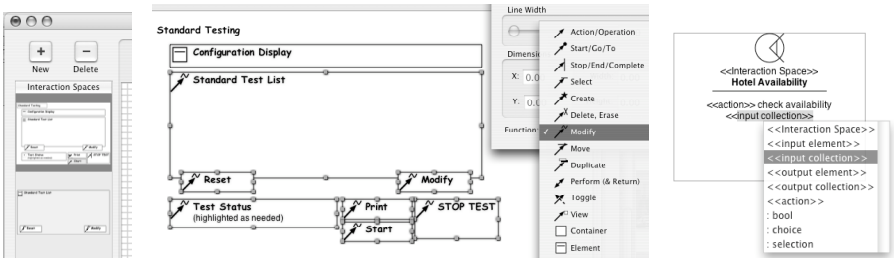

Fig. 4. Some of the developer-centered features in CanonSketch.

The center image on Figure 4 shows a selection of several canonical components to apply a transformation of their interactive function all at once. The rightmost image shows code completion for when the designer is specifying a Wisdom Interaction Space (which is a UML class stereotype representing ”space” where the user can interact with the application). We believe this way of editing UML models is more usable

than filling in complex forms that only update the UML view after validating everything the developer introduced.

Finally, the grid layout option may help position and resizing the components more rapidly, and the tool palettes follow the pattern of the successful Interface Builders. Tabbed-view navigation is important in order to achieve, in the future, model linkage at the various stages of the process.

4.2 A proof of feasibility: generating HTML forms

There is a third view in CanonSketch where a concrete prototype, in HTML form, is automatically generated, thus illustrating one possible concrete implementation. The concrete prototype is fully navigational, since it is rendered using an embedded, fully functional web browser, as we can see in Figure 5.

In order to verify the richness of the notation developed by Constantine and colleagues, and also to support automatic generation techniques, still without a semantic model defined, we built a proof of feasibility that can be exemplified in Figure 5. The HTML form shown was automatically generated from the canonical specification illustrated in Figure 3.

The HTML clickable prototype is useful for rapidly testing the navigational structure of the specified interface. The tool can also generate a PDF printable version of the Canonical/Wisdom models, which can act as a means to document the development process and commit to design decisions made with the client.

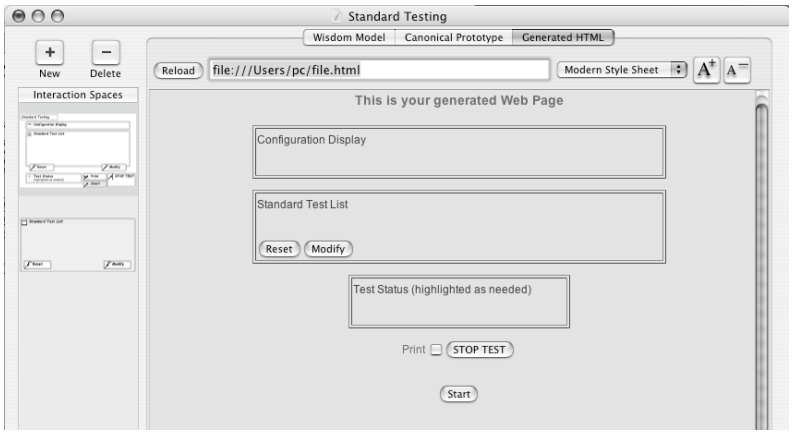

Fig. 5. Simple HTML automatically generated from the specification in Figure 2.

In the absence of a semantic model incorporated into our tool, this proof of concept already shows the potential of the notation, and achieves our goal of checking the richness of the abstract prototype notation. This is also part of our approach based on starting from a usable, simple tool and successfully add semantic mechanisms in an incremental way, rather than building a complex, formalism-centered tool.

5 Towards a Common Semantic Model

The automatic generation presented in the previous section was done at this stage without complete semantics of our intended adaptation of Canonical Abstract Prototypes. We are currently working on incrementally adding the mechanisms required to automatically generate concrete user interfaces from abstract prototypes.

From this initial proof of concept, we aim at specifying an extension to the UML 2.0 notation capable of fully supporting Canonical Abstract Prototypes. In particular, the Wisdom notation [7], which is a set of UML-compatible notations supporting efficient and effective interactive systems modeling, can be used and refined to achieve this goal.

In order to maintain synchronized Wisdom/Canonical views, a common semantic model is required. Specifying such a model will lead to a tool capable of not only supporting the design process at several stages (from early design ideas to concrete implementation) but also complementing the weaknesses of one model with the strengths of the other. The designer will be able to choose between one model view and switch back and forth while maintaining coherence between the models.

To support the modeling of presentation aspects of the UI, the Wisdom method proposes the following extensions to the UML [8]:

- «Interaction Space», a class stereotype that represents the space within the UI where the user interacts with the all the tools and containers during the course of a task or set of interrelated tasks;
- «navigate», an association stereotype between two interaction space classes denoting a user moving from one interaction space to another;
- «contains», an association stereotype between two interaction space classes denoting that the source class (container) contains the target class (contained); The contains association can only be used between interaction space classes and is unidirectional.
- «input element», an attribute stereotype denoting information received from the user, i.e., information the user can operate on;
- «output element», an attribute stereotype denoting information displayed to the user, i.e., information the user can perceive but not manipulate;
- «action», an operation stereotype denoting something the user can do in the concrete UI that causes a significant change in the internal state of the system.

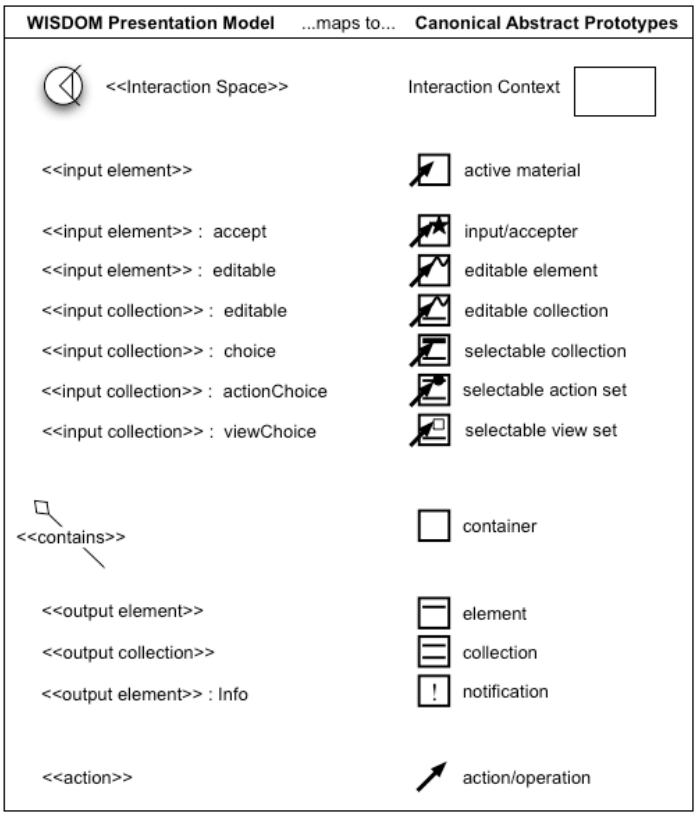

Fig. 6. Extending the Wisdom profile to support Canonical Abstract Prototypes: this figure shows the correspondence between Wisdom stereotypes and Canonical components.

Some problems identified with applying the Wisdom approach to UI patterns derive from the presentation aspects some of the patterns capture, such as size, position, or use of color [8]. Specifying a linkage between Canonical Abstract Prototypes and the Wisdom Presentation Model can help solve some of these problems, while also adding the necessary formalism to the Canonical notation.

In Figure 6, we show an initial specification of a possible connection between the Wisdom Presentation Model and Canonical Abstract Prototypes. An interaction space in Wisdom is clearly an interaction context in a Canonical Prototype.

Although not present in Figure 6, the «navigate» association can be unidirectional or bi-directional; the later usually meaning there is an implied return in the navigation. This essentially has the same meaning Constantine defines when describing the Canonical contexts' navigation map [1].

An «input element» attribute stereotype is mapped to a generic active material, unless typified. Input elements specify information the user can manipulate in order to achieve a task.

An «output element» attribute stereotype maps to an element and an «action» operation stereotype to an action/operation Canonical component.

The «contains» association stereotype is mapped to a Canonical container.

We can also see from Figure 6 that one possible initial extension to the Wisdom presentation model notation to fully support Canonical Abstract Prototypes consists in adding two more attribute stereotypes:

- «input collection», an attribute stereotype denoting a set of related information elements received from the user, i.e., a set of input elements; an «input collection» can be used to select from several values in a drop-down list, or choosing one element from a table to perform any given operation;
- «output collection», an attribute stereotype denoting a set of related information elements displayed to the user, i.e., a set of output elements. Typically, an «output collection» conveys information to the user about a set of elements of the same kind, for instance a search results list or the results display from a query to a database.

By typifying these attribute stereotypes, one can map a Wisdom presentation model to all Canonical components that belong to the classes of Materials or Hybrids. For instance, an input collection typified as choice can be mapped to a selectable collection. The designer starts by specifying the general structure of the UI using a UML extension (the Wisdom notation). That specification is mapped to one or more Canonical interaction contexts, where the designer expands and details the model in terms of size, position and interactive functions.

Figure 7 shows an example of a Wisdom Presentation Model for a Hotel Reservation System (described in and taken from [7]). Figure 8 depicts a Canonical Abstract Prototype that corresponds to the area inside the dashed rectangle in Figure 7. This mapping clearly shows the role of Wisdom interaction spaces realizing the interface architecture, and how it can be combined with the Canonical notation to help bridge the gap between abstract and concrete models of the user interface.

The capability of identifying UI patterns and expressing the solution in an abstract way independent of any particular platform or implementation is becoming more and more important, with the increase in the number of information appliances [8]. The Wisdom notation enables an abstract definition of UI patterns [8], and also complies with the UML standard. However, some problems remain for patterns expressing more concrete presentation aspects, such as size or positioning.

Having a tool that provides a common semantic model linking Canonical components to Wisdom elements can help solve some of these problems. It also adds the required formalisms for generating concrete user interfaces from Canonical specifications. We expect to incrementally build such a tool from our current version of CanonSketch.

As we will see in the next section, both notations can be used in conjunction in order to express abstract design patterns.

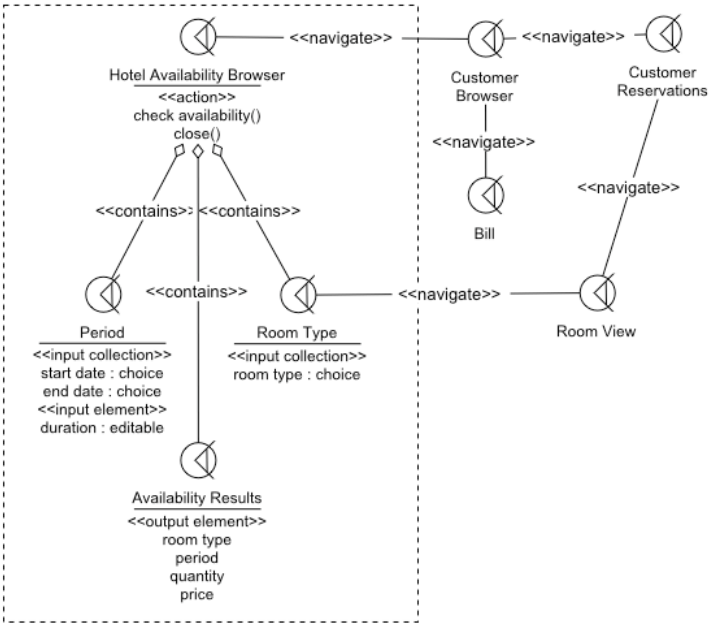

Fig. 7. A Wisdom Presentation Model for a Hotel Reservation System (described in and taken from [7]).

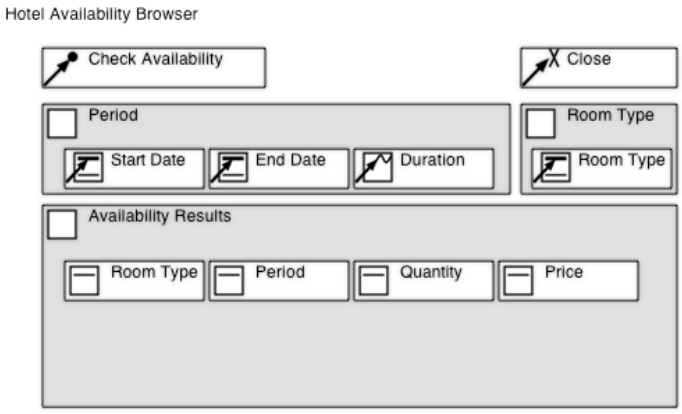

Fig. 8. A Canonical Abstract Prototype for the same Hotel Reservation System as in the area inside the dashed rectangle in Figure 7.

6 Using CanonSketch to represent UI patterns

Since the Canonical Abstract Notation is a way to express visual design ideas that was devised to support decision-making at a higher level of abstraction than concrete prototypes, we tried to investigate the ability to express GUI design patterns using CanonSketch. In this section, we present some examples of the Wisdom notation extension applied to some GUI patterns (taken from the Amsterdam collection [12]) and also the Canonical representation for the same patterns. As Constantine points out, "the ability to express design patterns in terms of generalized abstract models has seen little use in UI patterns". We still lack some widely accepted notation to represent commonly used solutions to some interaction tasks in an abstract way that can be applied to many design scenarios [8].

Throughout this section, all the Figures illustrate a Final User Interface (FUI) linked to a Concrete User Interface (CUI) or Abstract User Interface (CUI), in the terms defined in [16]. The FUI is represented by a screenshot of a particular implementation of the pattern, and the AUI is represented by the Canonical and Wisdom representations.

In Figure 9, we present the Wisdom and Canonical representations for the GUI Preview pattern [12]. We also present a concrete realization of this pattern (a dialog from MS PowerPoint). The problem this pattern tries to solve occurs when the user is looking for an item in a small set and tries to find the item by browsing the set. This pattern is particularly helpful when the items' content nature does not match its index (e.g. a set of images or audio files are indexed by a textual label). The solution is to provide the user with a preview of the currently selected item from the set being browsed [12].

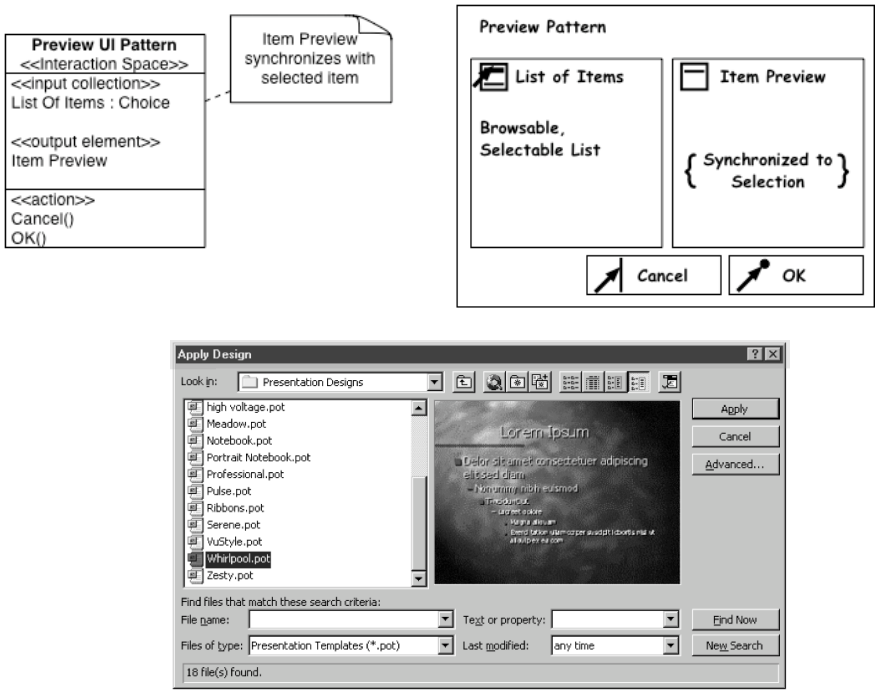

Fig. 9. Wisdom (bottom right) and Canonical (bottom left) notations applied to the Preview GUI pattern. A concrete example is shown on top: a dialog from MS PowerPoint.

As we can see, there is not much difference in this case. On the one hand, the Wisdom representation (on the right), is much more compact, because it is based on the UML. But the Canonical representation has the advantage of clearly stating that the browsable list of items is placed to the left of the item preview, which conforms with the western way of reading and therefore adjusts to the task being performed: the user first selects an item, and only then he focuses on the preview. It is also evident that the Canonical notation is much closer to the concrete representation of this pattern (on the top of Figure 9).

In the following pattern, the advantages of combining both Wisdom and Canonical representations are also evident. The grid layout pattern, also from the Amsterdam collection [12], tries to solve the problem of quickly understanding information and take action depending on that information. The solution is based on arranging all objects in a grid using a minimal number of rows and columns, making the cells as large as possible [12]. The top of Figure 10 shows an example of a concrete GUI where this is achieved (a dialog box from Word 97). By using this pattern, screen clutter is minimal and the layout is more consistent. The bottom of Figure 10 shows the Canonical representation at the left and the Wisdom UML representation on the right.

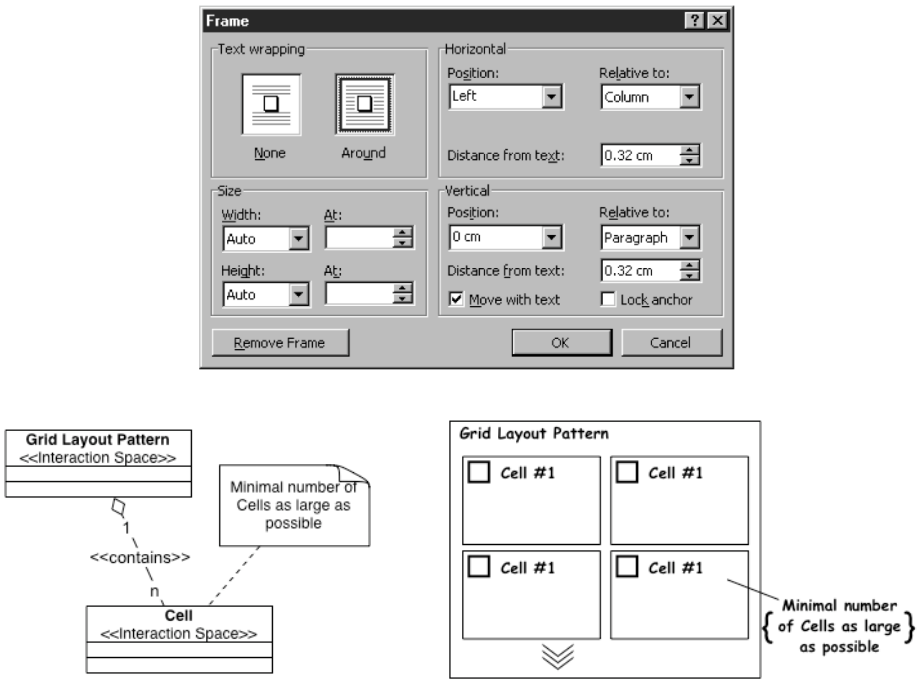

Fig. 10. The grid layout pattern: a concrete GUI application (top) and a Canonical (bottom left) and Wisdom (bottom right) representation.

It is clear that the Canonical notation has potential for easily expressing patterns that employ spatial, layout or positioning relationships between UI elements. Both notations have mechanisms for adding useful comments and constraints. The repetition element in the Canonical notation (represented by a triple chevron) is expressed as a one-to-many «contains» association in Wisdom.

Figure 11 shows a UI pattern where one can see the advantage of Wisdom over CAP. The "Wizard" pattern solves the problem of a user that wants to achieve a single goal, but needs to make several decisions before the goal can be achieved completely, which may not be know to the user [12]. Figure 11 shows an instantiation of this pattern through a Wisdom model (top left) that has two interaction spaces: Wizard body and 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*) [8].

This example illustrates an advantage of Wisdom over CAP regarding the modeling of navigation relationships between the abstract interface elements. In CAP, it is not possible to model a container that allows navigation to other instances of itself (like the Wizard step in this example). Modeling a containment relationship (like a Wizard body that contains successive interaction Wizard steps) is also difficult, unless an informal annotation or comments are used.

Finally, we show yet another abstract design pattern, the Container Navigation pattern [17]. When the user needs to find an item in a collection of containers, this pattern splits a window into three panes: one for viewing a collection of containers, one for viewing a container and one for viewing individual items. Figure 12 shows a Wisdom UML model, the Canonical prototype and a concrete GUI example of this pattern (Netscape's mail/news viewer).

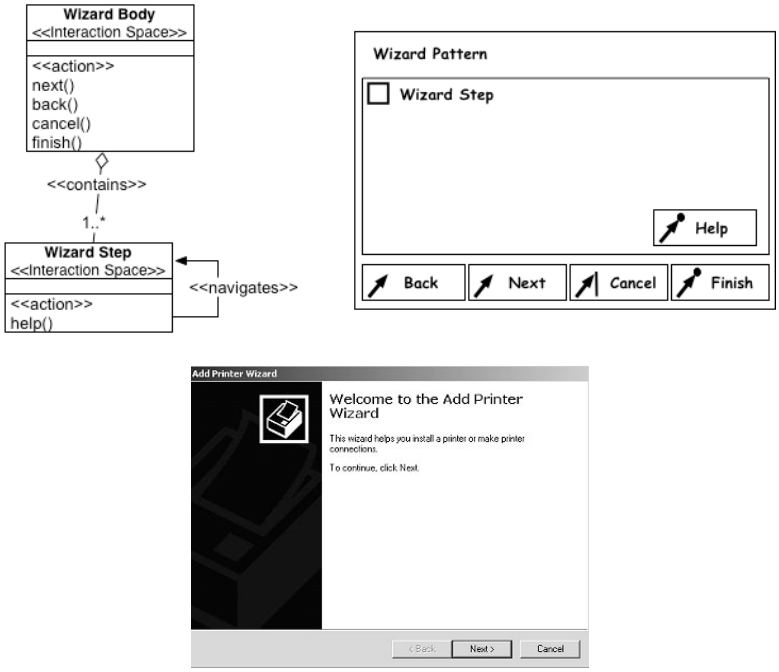

Fig. 11. The "Wizard" pattern. The top left part of the figure shows the Wisdom UML representation, which shows the navigation between "Wizard steps". The top right shows the Canonical representation and at the bottom a particular realization: the Add Printer Wizard in Windows 2000.

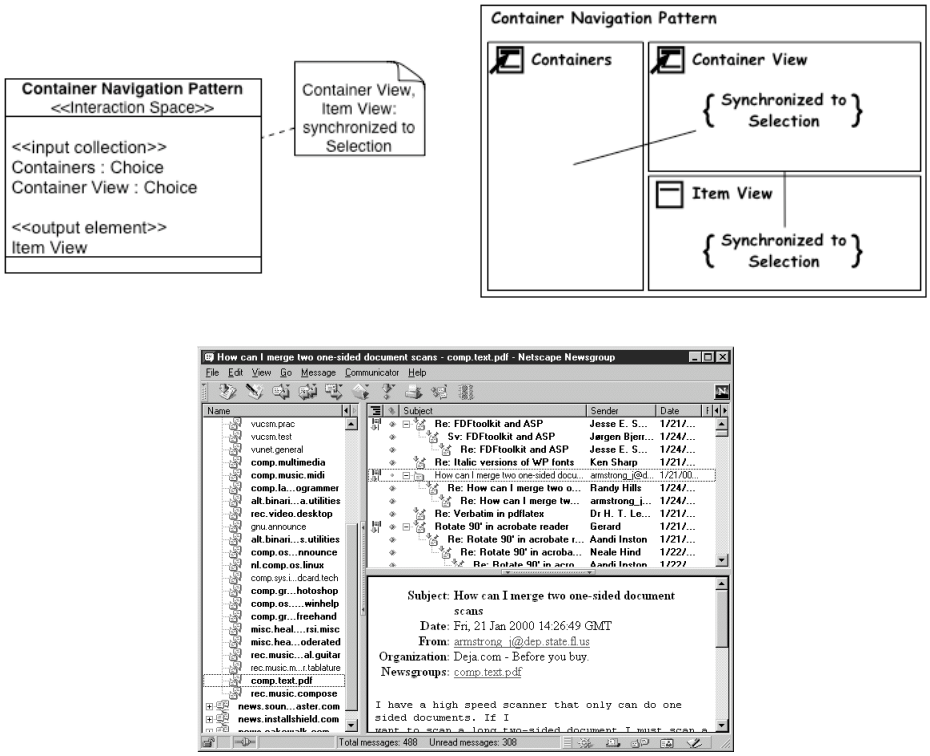

Fig. 12. The container navigation pattern: a Wisdom (top left) model, a Canonical prototype (top right) and a concrete GUI application (bottom), in this case Netscape's news reader.

In order to adequately express this UI pattern, size and relative positioning do matter. They support the user's task because the user first selects a container, then selects the item in the container and finally browses through the selected item. The information that the collection of containers occupies the left part of the screen, and that the item view is at the bottom right can only be conveyed through the Canonical notation.

To conclude, we observe that the Wisdom notation has some advantages over CAP, mainly due to its' compactness and the fact that is based on a language (UML) well understood and adopted by the majority of developers and designers. For expressing navigation patterns that involve several interaction spaces, such as the Wizard pattern [8], the Wisdom notation is more expressive and intuitive. Patterns dealing with spatial layout and size aspects are more clearly represented using CAP. The designer's mind works at several levels of abstraction, thus there is a need for languages and tools supporting those multiple levels of abstraction, while also maintaining a low learning curve.

When trying to express and compare the abstract design patterns presented in this section, we found CanonSketch to be a very useful and practical tool, because it supports two different notations that employ different levels of abstraction and also be-

cause it can easily be used to compile a collection of design patterns, thus simplifying the design's comparison and communication.

7 Conclusions and Future Work

To offer software engineers a usable, efficient and effective set of tools and methods is an important step towards building valuable, easy to use software. The same concepts that apply to the production of usable software also apply to the production of modeling tools. Our remit with CanonSketch is to achieve a modeling tool for MB-UID that is as easy to use as a drawing application. In this paper we presented the CanonSketch tool that supports the design of Canonical Abstract Prototypes as well as Wisdom Presentation Models. The CanonSketch project described here attempts to change the way modeling tools are built and envisioned. Existing tools are built using a formalism-centric approach, driven by the underlying semantics required by automatic generation techniques and not by the real needs of developers. Instead of focusing on the mechanisms required for automatic generation techniques, we focus on the successful features of usable software and on interaction idioms more closely related to Office-like applications.

One of the limitations of our approach is the fact that there is not a simple and clearly defined process of using the Canonical notation to specify interfaces for multiple devices. Although CanonSketch can clearly allow multi-platform development (Win, Mac, Palm, Web…) multimodal interfaces are not supported by this tool.

Nevertheless, even in the absence of model semantics, a tool like CanonSketch has significant value in specifying the architecture of complex interactive systems. Being able to generate HTML also means the notation is expressive enough to support automatic generation techniques and that it is possible to generate UI's for any platform based on GUI's and Forms like JavaSwing, Palm, Windows or MacOS. After this initial proof of feasibility, we presented a first specification for a UML extension based on the Wisdom notation that is a step towards a full support of Canonical Prototypes in a language that had a major impact on Software Engineering but still remains far from achieving the industrial maturity augured in the 90's, regarding UI modeling. We also showed how useful the tool can be in expressing UI patterns, and compared Wisdom UML representations of some patterns to the Canonical representations using the proposed correspondence between the two notations. We showed that patterns dealing with spatial or layout aspects could be adequately expressed in a Canonical representation, while Wisdom UML is better at modeling navigation relationships. We are currently finishing the integration of the semantic model of the UML into the tool. This will allow, among other possibilities, to export the abstract UI specification in XMI format, thus promoting artifact exchange between UML-based tools.

As for future work, it would be interesting to identify which notation designers prefer according to the development stage and the type of prototype they are busy with (low, mid or high fidelity). We also expect to refine the Wisdom notation taking advantage of the enhanced extensibility mechanism provided by UML 2.0, and add

other features such as knowledge management (capturing hidden information, like the most edited classes or interaction contexts, etc.), support for changing requirements and integration with application development in order to bridge the gap between industry and academy.

References

1. Constantine, L. and Lockwood, L. A. D.: Software for use : a practical guide to the models and methods of usage-centered design, Addison Wesley, Reading, Mass, 1999.
2. Constantine, L.: Canonical Abstract Prototypes for abstract visual and interaction design. In: Jorge, J., Nunes, N. and Falcão e Cunha, J. (eds.): *Proceedings of DSV-IS'2003, 10th International Conference on Design, Specification and Verification of Interactive Systems*. Lecture Notes in Computer Science, Vol. 2844. Springer-Verlag, Berlin Heidelberg New York, 2003.
3. Landay, J. and Myers, B.: Sketching Interfaces: Toward More Human Interface Design. IEEE Computer, pages 56-64, March 2001.
4. Myers, B., Hudson, S. and Pausch, R.: Past, Present and Future of User Interface Software Tools. *ACM Transactions on Computer Human Interaction*, 7(1):3-28, March 2000.
5. Newman, M., Lin, J., Hong, J. I. and Landay, J. A.: DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. Human-Computer Interaction, 18(3):259-324, 2003.
6. Nunes, N. J.: Wisdom - A UML based architecture for interactive systems. In *Proceedings of the DSV-IS'2000*, Limerick, Ireland. Springer-Verlag.
7. Nunes, N. J.: *Object Modeling for User-Centered Development and User Interface Design: the Wisdom Approach*. PhD Thesis, University of Madeira, Funchal, Portugal, April 2001.
8. Nunes, N. J.: Representing User-Interface Patterns in UML. In *Proceedings of OOIS'03 - 9th European Conference on Object-Oriented Information Systems*, pages 142-163, Geneva, Switzerland, 2003.
9. Nunes, N. J. and Campos, P.: Towards Usable Analysis, Design and Modeling Tools. In *Proceedings of the IUI/CADUI'04 Workshop on Making model-based UI design practical: usable and open methods and tools*, Funchal, Portugal, January 2004.
10. Trætteberg, H. Dialog modelling with interactors and UML Statecharts - A hybrid approach. In *Proceedings of DSV-IS'2003, 10th International Workshop on Design, Specification and Verification of Interactive Systems*. Springer-Verlag, 2003.
11. Trætteberg, H., Molina, P. J. and Nunes, N. J. (eds.): *Proceedings of the IUI/CADUI'04 Workshop on Making model-based user interface design practical: usable and open methods and tools*, Funchal, Portugal, 2004.
12. M. van Welie and Trætteberg, H.: Interaction Patterns in User Interface. In PLoP 2000. 2000.
13. Koch, N. and Wirsing, M.: Software Engineering for Adaptive Hypermedia Systems. In Paul de Bra, editor, *Third Workshop on Adaptive Hypertext and Hypermedia, 8th International Conference on User Modelling*, July 2001.
14. Schwabe, D. and Rossi, G.: *An Object-Oriented Approach to Web-Based Application Design*, Theory and Practice of Object Systems 4 (4), 1998. Wiley & Sons, New York.
15. Garrido, J. L. and Gea, M.: A Coloured Petri Net Formalisation for a UML-Based Notation Applied to Cooperative System Modelling. In *Proceedings of DSV-IS'2003, 10th*

International Workshop on Design, Specification and Verification of Interactive Systems. Springer-Verlag, 2003.
16. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces, Interacting with Computers, Vol. 15, No. 3, June 2003, pp. 289-308.
17. Nilsson, E. Combining compound conceptual user interface components with modeling patterns: a promising direction for model-based cross-platform user interface development. In *Proceedings of DSV-IS'2003, 10th International Workshop on Design, Specification and Verification of Interactive Systems*. Springer-Verlag, 2002.