Chapter #

FAST HI-FI PROTOTYING BY USING IDEALXML

A task-based approach to user interfaces design

Francisco Montero, Víctor López-Jaquero
Laboratory on User Interaction & Software Engineering (LoUISE)
Instituto de Investigación en Informática (I3A)
University of Castilla-La Mancha, 02071 Albacete, Spain
{ fmontero | victor }@info-ab.uclm.es

Abstract: Task modeling has become one of the cornerstones of model-based user interface design. Although different task modeling approaches to user interfaces design have been pushed, ConcurTaskTrees notation is becoming a *de facto* standard in the design of user interfaces including task-based modeling techniques. In this paper, a task-based approach to user interfaces design is introduced inspired by ConcurTaskTrees (Paternò, 1999). This approach is supported by a tool, namely IDEALXML, that allows for the animation of the specified user interfaces to generate a *hi-fi* prototype of the future user interface while still in the first development stages.

Keywords: Specification animation, task modeling, model-based design, user interfaces design tools.

1. INTRODUCTION

The main idea underlying in model-based approaches is identifying useful abstractions that highlight the main aspects of the design of an application. Model-based approaches for User Interface (UI) development have the potential to accommodate the increasing complexity of today's interactive applications. However, the mainstream developer has not adopted the model-based approach for creating UIs due to certain limitations (Myers et al., 2000).

The UI development task is one of the main design challenges in the creation of an application, since it must support the system's acceptance and be accessible and usable for everyone. Involving the users from the very beginning in the design process and focusing on usability, and not just on

technology, designers have tried to address this difficult challenge. To ensure the interface suited the target population, User-Centered Design (UCD) methods were introduced. On the other side, Usage-Centered Design (Constantine et al., 1999) partakes of the broadly UCD philosophy, but it emphasizes the fact that the center of attention should be the usage rather than the users per se. In the so called usage-centered design the attention is driven to those particular aspects of users that are more relevant to user interface design, fostering the linkage to use cases as a task or usage model.

Nowadays, software engineers use rapid prototyping to discover requirements by analyzing the prototypes built early in the development process and gathering feedback.

In this paper, we address fast *hi-fi* prototyping within a model-based UI environment. This approach is supported by a powerful visual tool, namely IDEALXML (Montero et al., 2005). UI design following the proposed approach is driven by task and domain models using a seamless mapping technique.

This paper is organized in three sections. First, an overview of model-based UI generation is presented. Next, UI description languages are introduced, focusing on usiXML (Limbourg et al., 2004). Finally, our approach to fast *hi-fi* prototyping is described.

2. USER INTERFACE GENERATION

The model-based approach was introduced to identify high-level models in order to allow the specification and analysis of an interactive system from a more semantic-oriented level, rather than dealing immediately with low-level implementation issues (Paternò, 1999). Unfortunately, the creation of the various models and the process of linking those models to each other is a tedious and time–consuming activity. Tools are required to attempt to relieve or hide these shortcomings.

In a model-based approach, the UI design is the process of creating and refining the set of models that describes the UI. In other words, model-based design focuses on finding the mappings between the various models (Vanderdonckt et al., 2003; Montero et al., 2005). Many facets as well as related models exist in order to describe the UI. A series of declarative models, such as domain, task, dialog, and presentation are interrelated to provide a formal representation of an interface design (Puerta, 1997) that finally will drive the generation of the UI.

Nowadays, we can find proposals in the literature that provide frameworks that enable UI development. At the beginning, most of those proposals would generate the UI out of a domain model. However, currently

most approaches drive their development out of a task model. Some of these proposals will be introduced in the next sections, describing the pros and cons of both domain-based UI design and task-based UI design.

2.1 Domain-Based Generation of User Interfaces

Domain model encapsulates the important entities of a particular application domain together with their attributes, methods and relationships. In particular, it captures concepts, objects and operations describing the domain. Within the scope of UI development, it defines the objects that the user requires in order to carry out his tasks.

Elements in the domain model possess attributes that are often relevant to UI presentation elements selection. Examples for these attributes are the data type, the range, the minimum and maximum value, etc. During the transformation process that generates the final UI, mappings are established which define, for example, which widgets should be used to display the value of an integer-type object for an input interaction task.

A study of the matching between the domain model elements and the UI ones has resulted in the following observations, which are enumerated bellow:

- Most of the main menu entries or navigation tabs correspond to one important class in the domain model
- There are some small help classes that are not presented in the UI at all.
- Text, integer and date attributes are represented by a static text label or a editable text field.
- Attribute types having a predefined number of values, are represented by an option-button or by an editable text-field that performs a syntax check.
- Singular references/pointers to other model classes are represented by a number of widgets showing important properties of the referenced object and a button leading to another visualization space showing additional information.
- Collections of references/pointers to other classes instances are represented by a list-view, having a column for each attribute.

These observations showed that important classes in the domain model are very visible in the UI, that class attributes often are a good indication of what data should be displayed and how it could be displayed, and that relationships between classes in the domain model are also represented in the UI.

Meaningful examples of this strategy in UI generation out of different types of domain models are Janus (Balzert, 1996), OlivaNova (Molina et al., 2004), Teallach (Griffiths et al., 1999) for desktop application, in web-based

environments WebRatio (Ceri et al., 2000) and VisualWade (Gómez, 2004) and in hypermedia applications OHDM (Schwabe et al., 1995).

These domain-based UI generation approaches produce complex UI, because users can see many elements at the same time. Moreover, as long as the user-task are not contemplated the dialog within the UI is rather limited and constrained, producing UI quite static.

2.2 Task-Based Generation of User Interfaces

Task model specifies what the user does, or wants to do, and why. It describes the tasks that users perform using the application, as well as how those tasks are related to each other. In other words, it captures the user tasks and the system behavior with respect to a task-set. ConcurTaskTree (CTT) (Paternò, 1999) is a well-accepted notation in the UI research development community used for the specification of task models. Despite the many advantages of CTT, large interactive systems described by using that notation can become too complex to be easily understood and can be really tedious to build (Paternò, 2001).

Most model-based development approaches define a dialog model by using a task model. Information from the task model is exploited in order to automatically or interactively derive the navigational structure of the application. In TERESA (Mori et al., 2004), structural information, as well as temporal relationships, are exploited in order to generate a so-called activation set. This set is later used to automatically generate the dialog model and the widgets of presentation model.

Our proposal is based on CTT notation, but a different set of icons are used in our tool, IDEALXML (Montero et al., 2005), to represent the different kinds of tasks. In Table 1 you can find the set of icons for the different types of tasks used:

Type of task	icon
abstraction	
application	
interaction	
user	

Table 1. Types of tasks and icons used in IDEALXML

Task-based design as opposed to domain-based one incorporates information regarding the tasks the user will carry out through the UI as well as the temporal relationships between those tasks. This kind of information allows addressing usability aspects such as UI overload, presentation elements grouping, etc.

2.3 User Interfaces Prototyping

Presentation model represents the content and organization of the user interface needed to support the identified tasks, apart from its appearance and behavior. Presentation model could be also called abstract prototype, since it represents, in the abstract, the contents of a user interface and how these contents are organized into interaction contexts, that is, the contexts within which users interact with the system.

Some of the main drawbacks of model-based user interface development have been the unpredictability of the final results and the lack of techniques for the evaluation of the final user interface given a set of declarative models (Myers et al. 2000). To overcome these drawbacks, and some other ones, different techniques have been introduced into human computer interaction development methodologies. One of those techniques introduced is user interface prototyping.

Prototyping consists in the creation of a preliminary version of the future user interface (prototype) so that the user and the experts can find possible problems in the design of the UI, both from the functional and from the usability points of view. Prototyping techniques fall into two main categories: (1) *lo-fi* (low-fidelity) techniques: this family of techniques are mostly used in requirements analysis stage to validate the requirements with the user in user-centered approaches. *Lo-fi* prototyping helps you apply Fudd's first law of creativity: "*To get a good idea, get lots of ideas.*" (Rettig, 1994). Paper prototyping, storyboards, card sorting, wireframes or sketching are some of the techniques widely extended in lo-fi UI prototypes creation. The main advantage of this kind of techniques is how quick and cheap the prototype is built and how easily this prototype can be modified. In (Granollers, 2004) a deep review of these *lo-fi* prototyping techniques can be found. (2) *hi-fi* (high-fidelity) techniques: they are aimed at the creation of preliminary version of the UI with an acceptable degree of quality. This kind of techniques produce a UI prototype which is much more closer to the final future one.

Although paper is still the most widely tool used in prototyping, some other tools have been proposed to try to make prototyping faster, easier to change or more accurate. In this sense, sketching tools like SketchiXML (Coyette et al., 2005) or CanonSketch (Campos et al., 2004) try to replicate the facilities in paper prototyping into a computer. SketchiXML is able to create a sketch of the UI by interpreting the drawings the user makes in a writable surface, such as the screen of a tablet PC, later the sketched user interface can be saved into a UI description language, namely usiXML (http://www.usixml.org). CanonSketch, on the other hand, allows for the specification of an abstract UI in terms of a Canonical Abstract Prototype

(Constantine, 2003). A different point of view is pushed in UI Pilot (Puerta et al., 2005). This tool provides a environment where the designer can build a prototype based on wireframes, that describe what should be implemented for each screen/page. Although the tool is rather interesting for the communication between requirements analysts and the developers, it fails to provide a formal framework to allow the designer to test an ongoing UI development.

Hi-fi prototypes could be considered to be better than *lo-fi* prototypes, since they are closer to the final user interface the user will interact with. Nevertheless, a set of disadvantages have been identified (Rettig, 1994) in *hi-fi* prototypes that need to be overcome: (1) these *hi-fi* prototypes take longer to be created and changed, (2) the reviewers/evaluators tend to comment more on the look and feel than on usability or function issues, (3) as *hi-fi* prototypes take more time and effort to be created their developers are more reluctant to introduce any change, (4) these prototypes can rise expectations that might not be achieved in the final version, and finally (5) a single bug in a test can bring a testing session to a complete halt, that is to say, the prototype must be robust.

3. MODEL-DRIVEN DEVELOPMENT IN USER INTERFACES DESIGN

During the last years, software engineering community has introduced the concept of Model Driven Architecture (MDA) (Vanderdonckt, 2005) which we find, in terms of goals, has some similarities with the model-based approach in UI engineering. The main benefit of MDA is the clear separation of the fundamental logic behind a specification from the specifics of the particular middleware that implements it.

In our proposal, we use models precisely because they actually speed up development and help us to get to a better solution more quickly. Good models clarify design issues and highlight tradeoffs, so design issues can be resolved rapidly. Models also help us to deliver better and more robust systems. In this sense, abstract prototyping was devised because it was found that the sooner developers started drawing realistic pictures or positioning real widgets, the longer it took them to converge on a good design (Constantine, 2003). Abstract models are always much simpler than the real thing.

3.1 XML-based UI description languages

Nowadays, a series of models are used within MB-UID approaches to describe UI. These models need to be stored in a repository so that they can be manipulated by the different tools used during UI generation stages. In most cases these models are stored using an XML-based format. In (Souchon et al., 2003) a review of the most prominent XML-based UI description languages can be found. UIML (Abrams et al., 1999), XIML (Puerta and Eisenstein, 2002), DiaMODL (Molina et al., 2004) or UsiXML (Limbourg et al., 2004) are meaningful examples of these kind of languages.

UsiXML provides an abstract user interface model that represents a canonical expression of the renderings and manipulation of the domain concepts and functions in a way that is as independent as possible from modalities and computing platform specifies.

Abstract object	Icon
Container	
Component	

Facet	Icon
input	
output	
control	
navigation	

Table 2. Abstract interaction objects and facets in usiXML and icons used in IDEALXML

We are using the abstract UI specification proposed in usiXML because it provides a reduced set of elements that allow the description of an abstract UI in a platform and modality independent manner. In Table 2 the set of icons used within our tool to represent the different elements of the abstract UI are shown.

4. FAST GENERATION OF HI-FI USER INTERFACE PROTOTYPES

One of the advantages of using a formal modeling language to specify the task model, such as ConcurTaskTrees, is the ability to simulate the system before it is built. Simulation can help to ensure that the system that is built will match users' conceptual model as well as to help to evaluate the usability of a system at a very early stage. Several task models simulators have been built for ConcurTaskTrees. For example, in CTTE the designers can specify a task model, which can be simulated. In IDEALXML designers can specify a task model and simulate the UI derived from the designed task model in an abstract manner by using CTT, usiXML and a set of heuristics

to transform the task model specification into an abstract UI. Currently, these heuristics are hardcoded in IDEALXML application code, but there is an ongoing work to support the use of transformation rules that the designer can modify following approach similar to the one proposed in (Limbourg et al., 2004).

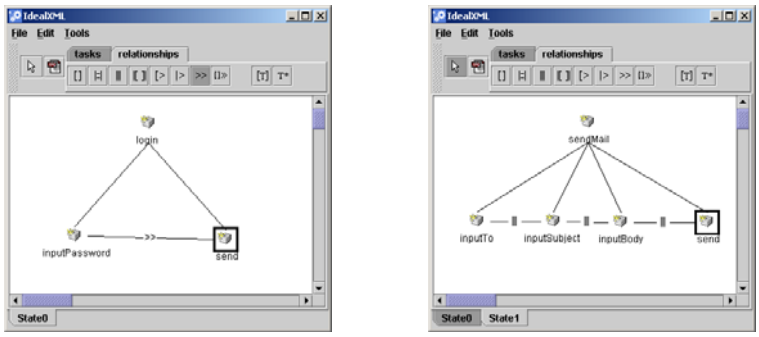

Figure 1. Task model specification in IDEALXML for e-mail sending task

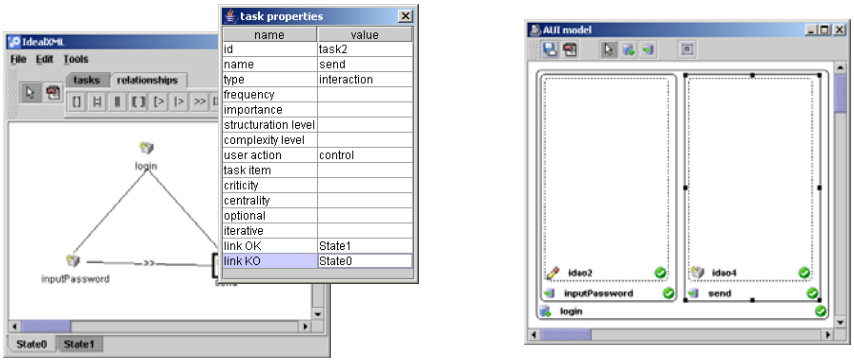

Figure 2. Abstract UI specification out of task model

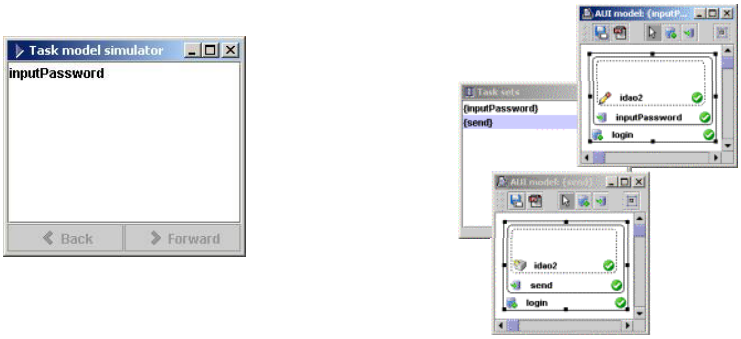

Figure 3. Simulation, ETS and abstract UI specification are available in IDEALXML

4.1 Abstract User Interfaces Prototyping

The previously mentioned hardcoded transformation rules are gathered in this section. It is fairly simple, straightforward rules govern transformations.

- Each cluster of interrelated task cases becomes an interaction space in the navigation map, so an abstract task is a container.
- A container also can be an interaction task or an application task if any of them are leaf in a hierarchical task decomposition.
- A component rises when we found an interaction or application task in a hierarchical task decomposition.
- A component can have several facets (input, output, control and navigation). These facets allow to the user interact with a system.

These transformation rules can be shown in Table 3.

Task model	*is an*	**Abstract presentation model**	
abstract task	*is a*	container	
interaction task	*is a*	*is leaf*: component	*input* *output* *control* *navigation*
		not leaf: container	
application task	*is a*	*is leaf*: container	
		is leaf: component	*output* *navigation*

Table 3. From task model to abstract presentation model

4.2 Abstract User Interfaces Prototypes Animation

IdealXML supports the animation of the abstract user interface resulting from the designed task model. This animation is grounded in the identification of the enabled task set (ETS) (Paternò, 1999). The ETSs for a specific task model is referred to as an enabled task collection (ETC).

Having identified the ETC for a task model, the next step is to identify the effects of performing each task in each ETS. The result of this analysis is a state transition network (STN), where each ETS is a state and transitions occur when tasks are performed.

In our proposal, the task model specification is split into states. Each state is a set of interrelated tasks, including temporal relationships between those tasks, usually connected to a *essential use case* (Constantine et al., 1999). In Fig. 1 the task model for sending an e-mail message can be found. Two

states have been identified in this case. The first one is related to user identification in the mail server and the second one is related to sending the e-mail message. By splitting the task model into states the task model complexity is drastically reduced and the legibility is really boosted.

States are connected by establishing links between them. Two different kinds of links are proposed *linkOK* and *linkKO*. *LinkOK* specifies which state the system should go to when the goal of the current state is successfully achieved. In a similar manner, *linkKO* is state the system should go to when the goal of the current state fails. For example, in Fig. 2 *linkOK* points to the state where the user can send the e-mail (it means that the user password provided was successfully validated) and *linkKO* points to current state (identification state, because the verification of the user password provided failed).

As in CTTE the designer can simulate task model specification in a textual manner, see Fig. 3a. In IDEALXML the designer is allowed also to animate the specification in a visual manner interacting with the abstract user interface. Moreover, at any time designers can select any set of tasks in the task model and get the abstract UI specification for the selected task in a graphical manner.

5. CONCLUSIONS

A good user interface design is essential to ensure the acceptance of a new software. It is a complex subject, but we can overcome this complexity by raising the level of abstraction in the design by using models. As long as models are aimed at working at an abstract level a mechanism is required to validate the design.

Abstract prototyping is a way to avoid the seduction of attractive prototypes that disguise weak designs. By making better use of modern visual development tools, abstract prototyping can speed up and simplify the design of highly usable systems and help us to produce improved and more innovative software products. In our fast abstract prototyping proposal we address most of the *hi-fi* prototypes shortcomings identified in (Rettig, 1994), providing an environment that allows the creation of the prototypes quickly in an abstract level enough to avoid focusing more on look & feel than in functional or usability issues and providing prototypes that can be easily modified.

Because the reduced set of elements used in usiXML to describe the abstract user interface and the graphical notations that we have provided in IDEALXML for each element, it is easy for the user to learn the notation and provide useful feedback to the designer.

6. ACKNOWLEDMENTS

This work is in part supported by the Junta de Castilla-La Mancha Regional grant PBC-03-003 and the Spanish CICYT TIN2004-08000-C03-01 grant. Also, we gratefully acknowledge the support of the SIMILAR network of excellence (http://www.similar.cc), the European research task force creating HCI similar to human-human communication of the European Sixth Framework Program.

REFERENCES

Abrams, M. UIML: An Appliance-Independent XML User Interface Language, in Proceedings of WWW8, Toronto, Canada. 1999.

Balzert, H., Hofmann, F., Kruschinski, V., Niemann, C. The JANUS Application Development Environment - Generating More than the User Interface. CADUI 1996: 183-208

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers 15*, 3 2003, 289–308.

Campos, P. and Nunes, N. Canonsketch: a User-Centered Tool for Canonical Abstract Prototyping. In Proceedings of DSV-IS'2004, 11th International Workshop on Design, Specification and Verification of Interactive Systems. Springer-Verlag, 2004.

Ceri, S. Fraternali, P. Bongio, A.: "Web Modeling Language (WebML): a Modeling Language for Designing Web Sites". WWW9 Conference, Amsterdam, May 2000.

Constantine, L. L., Lockwood, L. A. D. Software for use. Addison-Wesley. 1999.

Constantine, L.: Canonical Abstract Prototypes for abstract visual and interaction design. In: Jorge, J., Nunes, N. and Falcão e Cunha, J. (eds.): *Proceedings of DSVIS' 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.

Coyette, A., Vanderdonckt, J., A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces, Proc. of 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact'2005 (Rome, 12-16 September 2005), M.-F. Costabile, F. Paternò (eds.), Lecture Notes in Computer Science, Vol. 3585, Springer-Verlag, Berlin, 2005, pp. 550-564

Eisentein, J., Rich, C. Agents and GUIs from task models. In proceedings of 7th ACM Conference on Intewlligent User Interfaces IUI 2002. ACM Press. New York, 2002.

Gómez, J. Model-Driven Web Development with VisualWADE. ICWE 2004: 611-612

Granollers, T. MPIu+a. Una metodología que integra la Ingeniería del Software, la Interacción Persona-Ordenador y la Accesibilidad en el contexto de equipos de desarrollo multidisciplinares, University of Lérida, Spain, July 2004.

Griffiths, T., Barclay, P., McKirdy, J., Paton, N., Gray, P., Kennedy, J., Cooper, R., Goble, C., West, A., Smyth, M. Teallach: A model-based user interface development environment for object databases. In *Proceedings of UIDIS'99*. IEEE Press. 86-96.

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López Jaquero, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). LNCS, Vol. 3425, Springer-Verlag, Berlin, Germany, 2005.

Molina, P. User interface generation with OlivaNova model execution system. Intelligent User Interfaces 2004: 358-359

Montero, F., López Jaquero, V., Lozano, M., González, P. A User Interfaces Development and Abstraction Mechanism. In *HCI related papers of Interacción 2004*. Navarro Prieto, Raquel; Lorés Vidal, Jesús (Eds.) Springer.Verlag, Germany, 2005.

Montero, F., López Jaquero, V., Vanderdonckt, J., González, P., Lozano, M.D., Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML. 12th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'2005), Newcastle upon Tyne, England, July 13-15, 2005. Springer-Verlag, Berlin, Germany, 2005 (in print).

Mori G., Paterno` F., Santoro C. Tool Support for Designing Nomadic Applications. Proceedings ACM IUI'03, pp.141-148, January 2003, Miami, USA.

Myers, B., Hudson, S. E., and Pausch, R. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.* 7, 1 (Mar. 2000), 3-28, 2000.

Paternò, F. Model-based design and evaluation of interactive applications. Springer. 1999.

Puerta, A.R. A Model-Based Interface Development Environment. IEEE Software, pp. 40-47, 1997.

Puerta, A.R., Eisenstein, J.: XIML: a common representation for interaction data . IUI 2002: 216-217, 2002.

Puerta, A.R., Micheletti, M., Mak, A. The UI pilot: a model-based tool to guide early interface design. IUI 2005. 215-222, 2005.

Rettig, M. Prototyping for tiny fingers. *Communication ACM* 37, 4 (Apr. 1994), 21-27, 1994.

Schwabe, D., Gustavo Rossi, G. The Object-Oriented Hypermedia Design Model. Commun. ACM 38(8): 45-46. 1995.

Souchon, N., Vanderdonckt, J., A Review of XML-compliant User Interface Description Languages. Design, Specification, and Verification of Interactive Systems (DSV-IS 2003), pp. 377-391, 2003.

Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. CAiSE 2005: 16-31. 2005.