A Model-Based Interface Development Environment

ANGEL R. PUERTA, Stanford University

Mobi-D is a highly interactive environment that represents all relevant aspects of interface design in tightly connected declarative models. It supports user-centered development and allows structured design from abstract objects like user tasks.

Most interface development problems can be traced to two sources: the need for user-centered design environments and the lack of software systems that support all major development stages. Developer-centered environments, which are typical of most systems, give ample support to using and managing widgets, organizing and arranging layouts, and testing prototype interfaces, but fall short of answering key questions such as how widgets in a given dialog box can be used to accomplish a particular user task. The designer is forced to rely on her own experience to answer such questions or to distill a solution from loosely connected documentation.

In this article, I describe Mobi-D[1,2] (Model-Based Interface Designer), a comprehensive environment that supports user-centered design through *model-based interface development*. In the Mobi-D paradigm, a series of declarative models, such as user-task, dialog, and presentation, are interrelated to provide a formal representation of an interface design. This contrasts to model-based *systems*, which use only one or two models in isolation and have no explicit notion as to how the various model elements are organized into an interface design.

0740-7459/97/$10.00 © 1997 IEEE

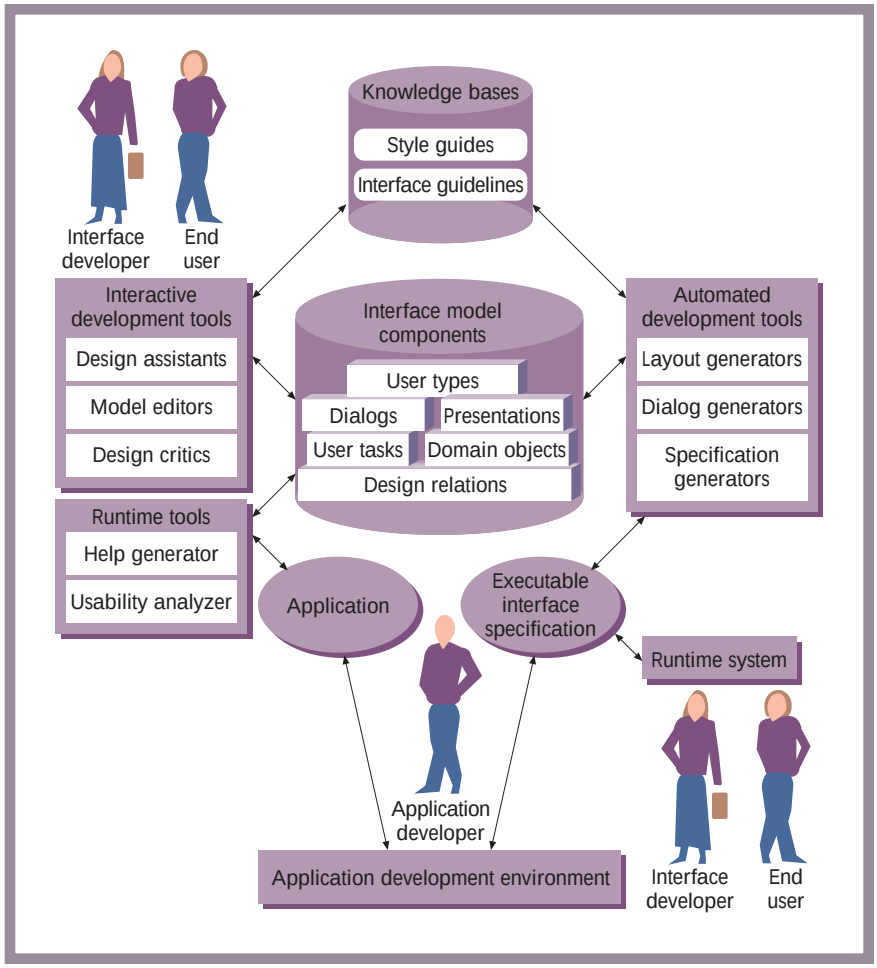

Figure 1. *Architecture of the Mobi-D environment for model-based interface development. Mobi-D emphasizes clear communication between the end user and interface developer, allowing the end user to participate in more aspects of development.*

Mobi-D offers a complete set of tools that support a clearly defined development life cycle. In model-based development, the interface is essentially an organized collection of interface objects. Design proceeds in a structured manner from abstract objects such as user tasks to the integration of tasks with domain elements and finally to an integrated interface ready for testing. Benefits include clearer communication throughout the development process, earlier involvement of the end user in key development decisions, and greater potential for reuse.

Mobi-D is an ongoing Stanford University project aimed at developing the next generation of model-based interface development software. It is a successor of the Mecano project,[3] a model-based system that automatically generates form-based interfaces from domain models. My group is currently using Mobi-D to design several interfaces in the medical and military logistics domains. Other groups at Stanford have used Mobi-D's tools and development cycle concept to reengineer existing interfaces or to evaluate prototypes.

We plan to eventually make Mobi-D available to researchers and developers in both industry and academia through a standard licensing process.

OVERVIEW

Mobi-D offers three key innovations to model-based interface development:

♦ *Metalevel modeling language*. Mobi-D uses one language to define the components, structure, and interrelations within interface models, as well as the elements of user interfaces. Consequently, all types of interface model elements can be explicitly interrelated, a client-server framework is feasible (with the interface model acting as the server), and runtime support tools can be linked to the models. All these are important goals of model-based environment research.

♦ *Formal definition of interface design*. Because design relations are identified as an explicit interface model component, formal definitions of interface design and related terms are possible, which the boxed text "Key Terms in Model-Based Development" provides. Interface design becomes the set of relationships among interface objects. It tells how, for example, a given widget relates to the model's dialog structure or presentation scheme, as well as to a domain element or to a user task.

♦ *Design philosophy of decision-making support*. Interface design supports the decision making of developers and end users, as opposed to just generating the interface automatically—the approach most existing model-based systems take.

Mobi-D also provides the following:

♦ *Continuous end-user involvement.* In many instances, user-centered design fails because of poorly understood or weak connections between the task-analysis and implementation phases. Mobi-D establishes clear communication between the end user and interface developer, as well as between the end user and developer and the interface design process itself. More important, this communication is grounded in a computational model with defined operators and software support. Thus, Mobi-D makes it easy to engage the end user in the development process, always keeping a rational view of how design decisions affect user tasks. The end user can more easily visualize how a given widget affects the completion of a task, for example.

♦ *Greater reuse potential*. Because complete interface designs are available in declarative form, they can be indexed, processed, and reused in other applications.

Architecture. Figure 1 shows the architecture of the Mobi-D environment. The *interface model* acts as a central repository of knowledge about an interface design. It is a declarative representation of all relevant aspects of a user interface, including its components and design. It typically

KEY TERMS IN MODEL-BASED INTERFACE DEVELOPMENT

Because one language is used to capture the interrelationships of the various models, it becomes possible to define a set of terms. These foundational terms make relationships explicit and help clarify communication among the interface stakeholders.

Interface model. An interface model is a declarative representation of all relevant aspects of a user interface, including the components and design of that interface. It typically embodies a number of interface objects at different levels of abstraction: user tasks, domain elements, presentations, dialogs, user types, and design relations. These objects are normally organized into models (user-task or presentation models, for example) within the overall interface model. Interface models are expressed via an interface modeling language.

Interface modeling language. A language that defines the organization, components, and relationships of interface models. It is used to build interfaces and interface designs.

Interface. An organized collection of interface objects.

Interface design. A set of relationships among interface objects in an interface. It answers the question of how, for example, a given widget relates to a dialog structure, to a presentation scheme, to a domain element, and to a user task.

Model-based interface development environment. A software environment that supports the creation of interface designs under a specific interface modeling language. Environment tools are typically grouped into design-time tools, runtime tools, and runtime systems. It is characterized by the connectedness of the models and components within it. It differs from a model-based interface development *system*, which may use one or possibly two models, but without an explicit definition of how the models are organized into an interface design.

Design-time tools. The set of software tools that operates on an interface model to build an interface design. They may be further classified into interactive tools (such as a model editor), or automated tools (such as a layout generator).

Runtime tools. The set of software tools that use an interface model to support end-user activities, such as generating animated help or collecting and analyzing usability data.

Runtime system. A system that takes an executable interface specification generated from an interface model and allows the previewing, running, and testing of the interface. In some model-based environments, these functions are provided in an interpretative fashion (the runtime system immediately reflects changes to the interface model). In others, recompilation is needed to update the executable interface specification.

embodies some number of interface model components: design relations, user tasks and domain objects, dialogs and presentations, and user types. The components are models (user-task or presentation, for example) within the overall interface model. The boxed text "How Model-Based Interface Development Evolved" describes how some model-based systems use one or two of these components as separate entities.

Developers access and modify the interface model through tools that offer a variety of functions and levels of automated support. In a model-based environment, tools are typically grouped into design-time tools, runtime tools, and runtime systems. *Design-time tools* operate on an interface model to build an interface design. They include *interactive development tools*, such as a design assistant, and *automated development tools*, such as a layout generator.

Runtime tools use an interface model to support end-user activities, such as generating animated help or collecting and analyzing usability data. A *runtime system* takes an executable interface specification generated from an interface model and allows various individuals (such as the developer and end user) to preview, run, and test the interface.

The tools may also use additional *knowledge bases* about design guidelines and principles to operate on the interface model or to advise the developer. An interface model may be transformed into an executable interface specification. This specification includes a coupling mechanism with application-specific code to deliver a final application.

Development cycle. Figure 2 shows the Mobi-D development cycle. All the processes are fully interactive and may involve the end user. The cycle is iterative: It begins with the elicitation of user tasks (on the left). An interactive Mobi-D tool lets the end user enter a textual task description. The tool elicits key terms such as objects and actions and guides the end user in editing and refining the terms into a structured user-task description. The developer uses this description to build the user-task and domain models in parallel with the help of Mobi-D's model-editing tools. He then integrates the models so that domain objects are related to the user tasks for which they are relevant.

The decision support mechanisms in Mobi-D use the user-task and domain models to recommend presentation and interaction techniques. Mobi-D displays these recommendations to guide the developer during design and to ensure that all task and data elements are embodied in the interface. In effect, Mobi-D walks the developer through the selection and layout of interface components, providing for each subtask a choice of optional components preconfigured for the task data. For example, if the user must enter a number, Mobi-D provides a choice of slider and text-entry widgets with labels and range bounds that are consistent with the model. The developer can select the slider, position it in the dialog window, resize it, edit its label, change its color, and so on. The end user then tests the resulting interface.

SAMPLE APPLICATION

To illustrate Mobi-D's capabilities,

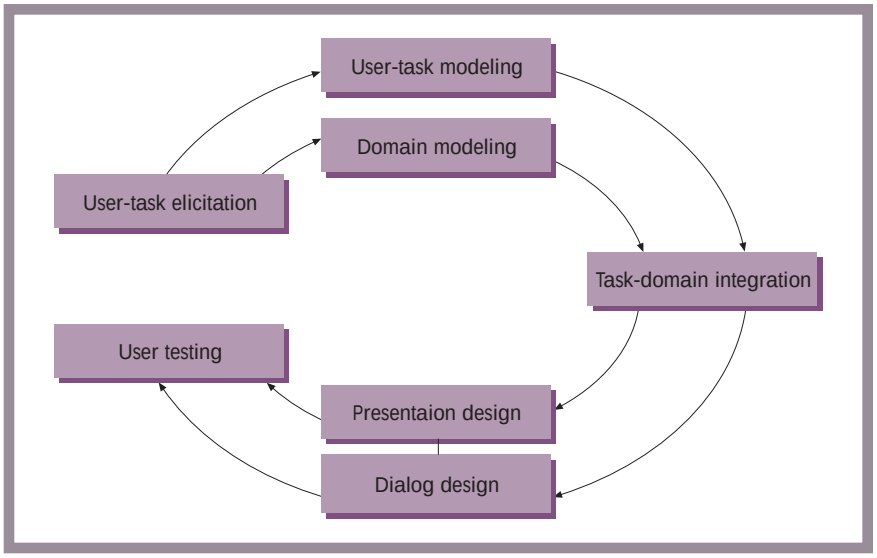

Figure 2. *The interactive, user-centered development cycle in Mobi-D.*

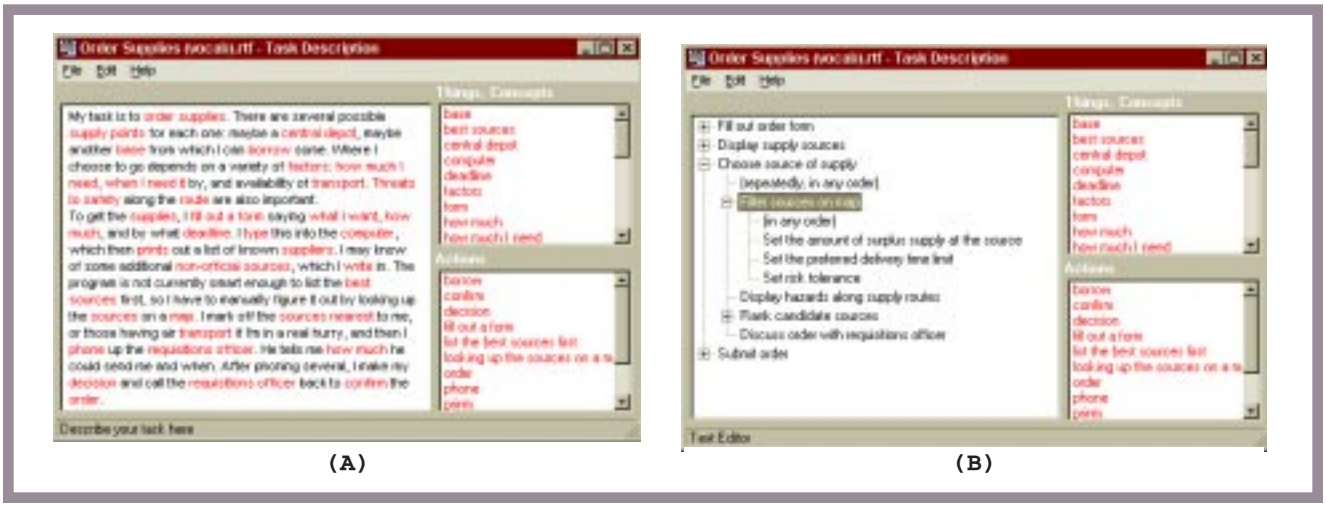

Figure 3. *Mobi-D's user-task elicitation stage:* ***(a)*** *an informal task description, followed by a list of key items (boxes at right) and* ***(b)*** *a formal task outline, generated with help from the developer.*

we used it to design an interface in the logistics domain. The interface is part of a system that will let users visualize and operate on logistics data and functions in a military theater of operations. As such, it must provide access to a distributed information space and be able to adapt to the needs of many different user types. The examples given here are from the development of a subset of the interface, which lets a requisitions officer select the best possible sources for a given supply. The officer views a list of available suppliers and ranks them according to a set of criteria, such as how much surplus each supplier has, how quickly it can deliver, and how much risk is involved. At any time, the officer can select a supply source and evaluate the transportation risks from source to destination. Once the officer selects a source, he places the order either through e-mail or by phone.

Eliciting the user tasks. One of the central activities of any user-centered design is the construction of a user-task model. This process requires close collaboration between an end user, or domain expert, and an interface developer. Mobi-D involves the end user directly in the development of user-task models.

Figure 3 shows two screens in the elicitation phase. In Figure 3a, the end user describes the task informally, not worrying how it may eventually be converted into a computer model. He then identifies key actions and things relevant to the task (boxes at right in Figure 3a). In Figure 3b, the end user, with the help of the developer, has worked out an outline of the user tasks.

The development of the task outline serves two purposes. First, it establishes an organized channel of communication between end users and developers. This reduces the chances of misunderstood or incomplete requirements. Second, it provides a software product that the developer can use directly in the next phase.

User-task and domain modeling. After a user-task outline is completed, the developer starts interacting directly with Mobi-D to create user-task and domain models. Mobi-D reads a skeleton of these models directly from the task outline and then, with input from the developer and possibly the end user, refines them by setting properties for each subtask and domain object. The user-task

HOW MODEL-BASED INTERFACE DEVELOPMENT HAS EVOLVED

Most early interface models had no computational equivalent.[1] They were used to define components and their functionality and guide the design of interfaces and interface development environments. L-CID is an example of a system based on an abstract interface model.[1] L-CID implements a model of an intelligent interface in a blackboard architecture. It was used to rapidly prototype machine-learning-based interaction techniques for user interfaces.

Data models. The data model was the first type of explicit model to appear in model-based interface development. It was borrowed directly from the data structures defined by the application and proved to be useful in generating the widgets of an interface by matching data types with widget types. Systems such as UIDE[2] and Don[2] used data models and applied some type of layout algorithm to produce complete static interface layouts.

However, developers could not obtain a specification of interface behavior from data models, so more expressive models were needed. Current model-based environments typically use data definitions as a subcomponent of their interface model.

Domain models. Advances in software engineering allowed model-based systems to move beyond simple data models. Thus, systems started to use entity-relationship data models (Genius[3]), enhanced data models (Trident[4]) or object-oriented data models (Fuse[5]). Eventually, these models led to fully declarative domain models, such as those in Mecano (see main text),that could effectively express the relationships among the objects in a given domain. As a consequence, it became possible for the first time to automatically generate partial specifications of dynamic interface behavior along with a static layout.

Application models. Though elaborate, domain models do not describe semantic functions in the application's functional core that are associated to objects in a domain. To address this lack, some model-based systems—including UIDE,[2] Trident,[4] and Humanoid[6]—introduce an application model in various forms. The goal of these models is to make it easier to declare interface behavior. The UIDE application model, for example, consists of application actions, interface actions, and interaction techniques. The developer assigns parameters, preconditions, and post-conditions to each action and then uses them to control the user interface at runtime.

Other partial models. Application and domain models are considered partial interface models because they do not cover all the relevant aspects of an interface design. Naturally, other partial interface models have emerged over the years. User-task, dialog, and presentation models are the most significant. Additional partial models, such as platform, user, and workplace models have been defined but have not been used much in practice and so have little software support. Instead many systems have subsumed the characteristics of these models into the presentation and dialog models.

User-task models. The user-task model is the most crucial in supporting a user-centered design philosophy. It describes the tasks an end user performs and dictates what interaction capabilities must be designed. The model typically involves elements such as goals, actions, and domain objects. Goals specify when a desired state is met, sequences of actions define procedures to achieve a goal, and domain objects represent elements that must be displayed in the interface to complete each task in the model. Adept,[7] Fuse,[5] Tadeus,[8] and Trident[4] embed various forms of user-task models.

The user-task model represents a significant advance in model-based development. It establishes a methodology for task-based design: the user-task model drives the generation of alternative design solutions to support the same interactive task. Adept provides an integrated design support environment for this methodology.

Dialog model. The dialog model describes the human-computer conversation. It specifies when the end user can invoke functions through various triggering mechanisms (push buttons, commands, and so on) and interaction media (voice input, touch screen, and so on), when the end user can select or specify inputs, and when the computer can query the end user and present information. Many dialog models have shown evidence of success: dialog nets (Genius,[3] Tadeus[8]), attributed grammars (Boss[5]), state-transition diagrams (Trident[4]), dialog templates (Humanoid[6]). However, no consensus of an ultimate dialog modeling technique has emerged.

Presentation model. The presentation model specifies how interaction objects (or widgets) appear in the different dialog states. It generally consists of a hierarchical decomposition of the possible screen displays into groups of interaction objects. By definition, presentation and dialog models are closely interrelated, which is why some model-based development environments consider them together: ITS[9] typically falls in this category by providing a style library. A style is a coordinated set of decisions on presentation and dialog used consistently throughout a family of applications.

Comprehensive models. Partial models have led to current efforts to define comprehensive interface models, like Mobi-D (described in the main article) and Mastermind.[10] This evolution has followed that of component development itself. As interface models have evolved from data models to comprehensive model-based environments, there has been a corresponding move from elementary toolkit library components to basic textual or graphical editors, to prepackaged elements such as ActiveX that can be used to construct interfaces piecewise. This confluence of comprehensive interface models and complex primitives is what gives model-based development its solid foundation. The technology would have limited applicability if it did not have rich interface models; if no suitable primitives were available, programmers would have to model at a very low detail level (almost at a programming level).

This foundation will serve as a basis for an increasing number of user-centered interface development environments that

will support a robust development cycle and a range of interface designs.

REFERENCES

1. A. Puerta, "The Study of Models of Intelligent Interfaces," *Proc. Int'l Workshop Intelligent User Interfaces*, ACM Press, New York, 1993, pp. 71-78.
2. J.D. Foley, "History, Results and Bibliography of the User Interface Design Environment (UIDE): An Early Model-Based System for User Interface Design and Implementation," *Proc. Eurographics Workshop Design, Specification, Verification of Interactive Systems*, F. Patern, ed., 1995, http://www.info.fundp.ac.be/~jvd/dsvis/dsvis94.html.
3. C. Janssen, A. Weisbecker, and J. Ziegler, "Generating User Interfaces from Data Models and Dialogue Net Specifications," *Proc. Conf. Human Factors in Computing Systems: InterCHI '93*, ACM Press, New York, 1993, pp. 418-423.
4. J. Vanderdonckt and F. Bodart, "Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection," *Proc. Conf. Human Factors in Computing Systems: InterCHI '93*, ACM Press, New York, 1993, pp. 424-429.
5. F. Lonczewski and S. Schreiber, "The FUSE-System: an Integrated User Interface Design Environment," *Proc. CADUI 96*, J. Vanderdonckt, ed., http://www.info.fundp.ac.be/~jvd/dsvis/cadui96.html.
6. P. Szekely, P. Luo, and R. Neches, "Beyond Interface Builders: Model-Based Interface Tools," *Proc. Conf. Human Factors in Computing Systems: InterCHI '93*, ACM Press, New York, 1993, pp. 383-390.
7. S. Wilson and P. Johnson, "Bridging the Generation Gap: From Work Tasks to User Interface Designs," *Proc. CADUI 96*, J. Vanderdonckt, ed., http://www.info.fundp.ac.be/~jvd/dsvis/cadui96.html.
8. E. Schlungbaum and T. Elwert, "Automatic User Interface Generation from Declarative Models," *Proc. CADUI 96*, J. Vanderdonckt, ed., http://www.info.fundp.ac.be/~jvd/dsvis/cadui96.html.
9. C. Wiecha et al., "ITS: A Tool for Rapidly Developing Interactive Applications," *ACM Trans. Information Systems*, July 1990, pp. 204-236.
10. P. Szekely et al., "Declarative Interface Models for User Interface Construction Tools: The Mastermind Approach," in *Engineering for Human-Computer Interaction*, L. Bass and C. Unger, eds., Chapman & Hall, London, 1995, pp. 120-150.

and domain models are the foundation of an interface design.

The decision support tools in Mobi-D provide recommendations for presentation and dialog design in accordance with the structure and properties of these models.

Figure 4a and 4b show views of the user-task and domain models in the corresponding Mobi-D editors. The editors are divided into three panes. The top left pane shows the current model; the bottom pane shows the properties of the selected object. The top right pane lists prototype objects available for use in the current model via a drag-and-drop operation. Developers can make an object from the current model a prototype by dragging it into the top right pane. Because prototype objects are organized as user-task or domain models, Mobi-D can save them apart from the current model for reuse in a future design.

Figure 4c shows the integration of the user-task and domain models. When the developer assigns a domain element to a specific user task, Mobi-D creates a design relation. The set of all design relations in an interface constitutes the interface design. Eventually, the user-task-to-domain-element relations expand to include presentation and dialog elements. Because Mobi-D interface models store these relations explicitly, developers and end users have a framework for rationally visualizing and reusing interface designs.

Presentation and dialog design. As Figure 2 shows, presentation and dialog designs occur in parallel in Mobi-D. Decision support tools examine the user-task and domain models, along with perhaps a set

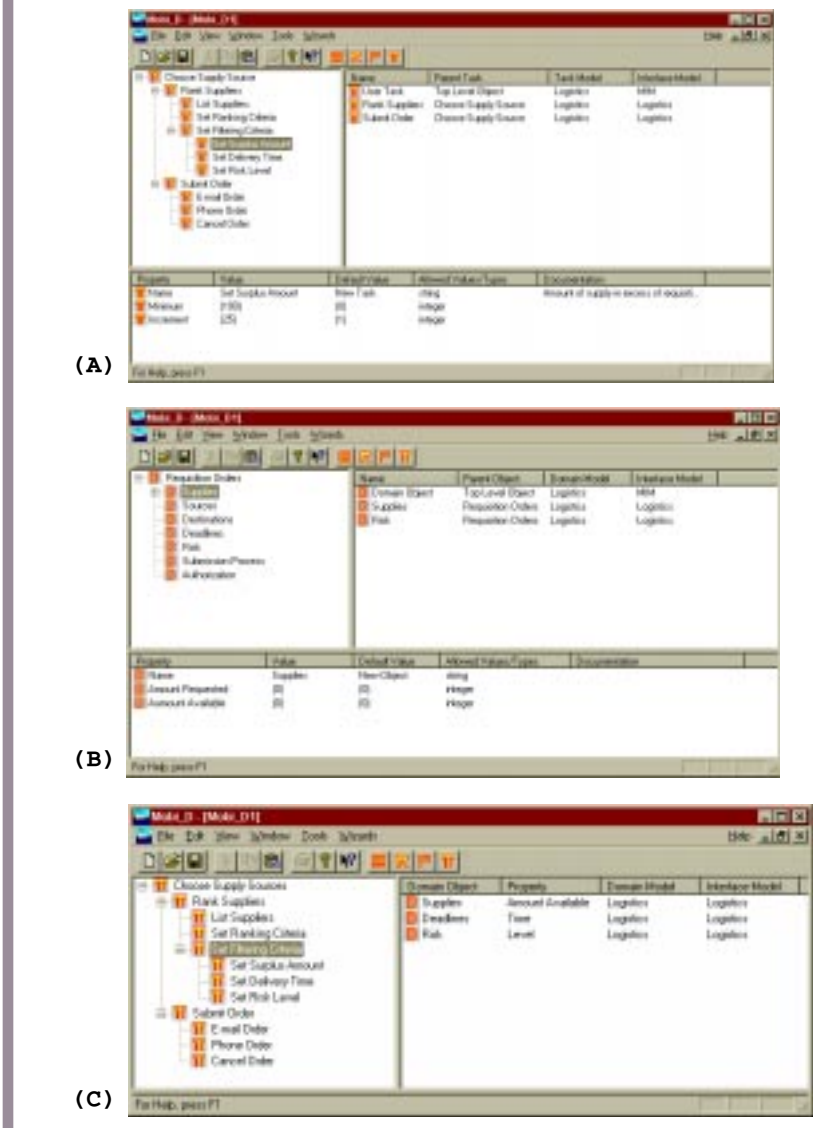

***Figure 4**. Model editing and integration in Mobi-D: **(a)** editing the user-task model, **(b)** editing the domain model, and **(c)** integrating the models.*

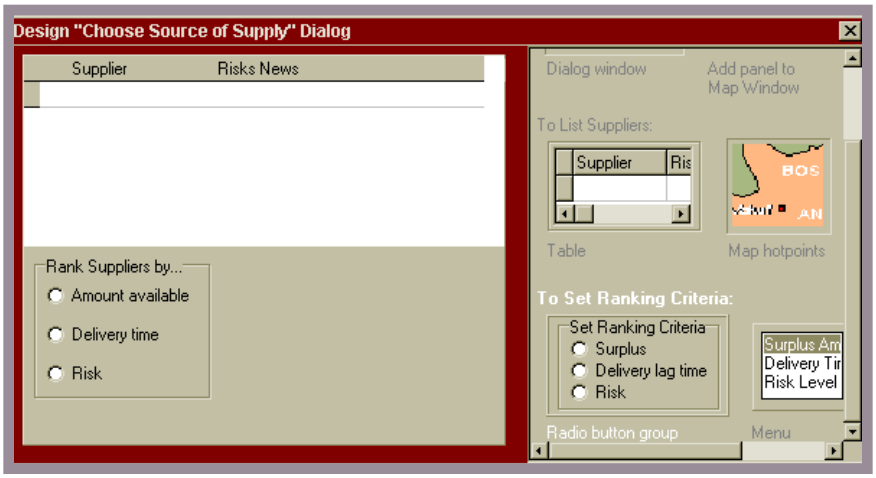

Figure 5. *A snapshot of the design of a presentation and dialog using Mobi-D.*

Figure 6. *The sample interface ready for user testing.*

of interface guidelines, and recommend the widgets, or interaction elements (interactors), that should be used to complete each subtask in the user-task model. The developer can override any of the recommendations made and choose different elements.

Figure 5 shows a snapshot of the presentation and dialog design process. The right side of the window depicts the palette of interactors that Mobi-D has arranged for the developer. The left side is the canvas, where the developer can drop and lay out selected interactors. Mobi-D steps the designer through each subtask in the user-task model and ranks available interactors according to guidelines and to the type of data that each interactor must display. The developer is free to choose any available interactor. By following this process, Mobi-D makes sure that each user task defined in the user-task model is appropriately displayed and completed in the resulting interface.

Because of the clear connection between presentation and dialog design and the user-task and domain models, Mobi-D makes it easier to obtain input and advice from the end user during this phase. Because the end user can better visualize how each interactor relates to the task outline built at the beginning of the development cycle, he can make more informed decisions and gain better insight into the effect each interface element has

on the overall task. When the interface design is completed (see Figure 6), the end user can test it and again provide feedback on the way the design relates to the constructed interface model.

Mobi-D is the most recent step in the evolution of model-based interface development systems. I expect this technology to continue to evolve. I also believe it will significantly affect the way interfaces, and consequently applications, are built in the future. This paradigm has the potential to influence several areas outside development life-cycle activities.

First, usability engineers can use interface models to support and integrate experiment design, data collection, and results analysis. They can precisely map the actions of an actual user during an interaction session to a user-task model. They can also correct deficiencies in usability more efficiently by examining the relations between user actions and the elements of an interface model. In this manner, usability processes can be integrated into the development cycle and offer software support.

Second, some model-based environments, like Mobi-D, support interface design by offering guidance, not by automating the process. As such they are actually CAD tools, advisers that assist interface designers in selecting design options by proposing accurate and reliable values that rely on design knowledge. The recommendations do not limit flexibility but rather organize decision making. Throughout the development cycle, the designer remains free to control each step by deciding which is the right value for each design option. Tedious or repetitious tasks, on the other hand, are automated so that efficiency is increased.

Finally, because the design environment must treat interface primitives as encapsulated objects with predefined functionality, model-based development systems are a good fit for Internet-based user interfaces that rely on elements such as Java applets or ActiveX controls. The modularity of those elements suggests a component-based framework for interface development that model-based systems can exploit to establish a methodology for distributed interfaces. ◆

ACKNOWLEDGMENTS

The work on Mobi-D is supported by the Defense Advanced Research Projects Agency under contract N66001-96-C-8525. David Maulsby contributed significantly to the design of the Mobi-D tools. I also thank Eric Cheng, Kjetil Larsen, Justin Min, and Chung-Man Tam for their work on the development of Mobi-D. Finally, I thank Egbert Schlungbaum, Jean Vanderdonckt, and Pedro Szekely for their comments and opinions on an earlier version of this article.

REFERENCES

1. A. Puerta, "The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development," *Proc. CADUI 96*, J. Vanderdonckt, ed., http://www.info.fundp.ac.be/~jvd/dsvis/cadui96.html.
2. A. Puerta and D. Maulsby, "Management of Interface Design Knowledge with MOBI-D," *Proc. Int'l Conf. Intelligent User Interfaces*, ACM Press, New York, 1997 (to be published).
3. A. Puerta, "Model-Based Automated Generation of User Interfaces," *Proc. Nat'l Conf. Artificial Intelligence*, MIT Press, Cambridge, Mass., 1994, pp. 471-477.

Angel Puerta is a research scientist at Stanford University, where he leads a team of researchers working on Mobi-D and where he developed Mecano. He has published numerous articles on model-based interface development and has an upcoming book on the subject from Kluwer Academic Publishers.

Puerta received a PhD in computer engineering from the University of South Carolina. He is a member of the steering committee for the annual conference on intelligent user interfaces and a member of the program committee of the annual Computer-Human Interaction conference. He is a member of the IEEE and ACM.

Readers can contact Puerta at Stanford University, MSOB x215, Stanford, CA 94305-5479; puerta@smi.stanford.edu. For additional information on Mobi-D, contact http://www-smi.stanford.edu/projects/mecano.