Supporting User-Centered Design of Adaptive User Interfaces Via Interface Models

Angel R. Puerta
Stanford University
251 Campus Drive – MSOB x215
Stanford, CA 94305-5479 USA
+1 650 723 5294
puerta@smi.stanford.edu
http://www.smi.stanford.edu/projects/mecano

ABSTRACT
Model-based development is an emerging technology for the design and specification of interfaces from declarative interface models. Key benefits of this technology are: (a) centralization of all knowledge and data for an interface design in a single representation: the interface model, (b) definition of a user-centered interface development software cycle, (c) architectural support for multiple types of design-time and runtime tools based on the interface model representation, (d) rational visualization of abstract and concrete elements of an interface design, and (e) a framework for the incorporation of decision support tools to aid the design process. In this paper, we introduce the key conceptual terms in this technology along with the architecture and the development cycle of Model-Based Interface Development Environments. Furthermore, we survey the state-of-the-art in the field, illustrate the application of the paradigm with an example from the logistics domain, and critically analyze the impact of the technology on creating effective user interfaces and on the future of software development.

Keywords
Model-Based Interface Development, Interface Models, User-Interface Development Tools

THE PARADIGM OF MODEL-BASED INTERFACE DEVELOPMENT

The User Interface is a central element of any modern application program, one that often determines how well end users accept, learn, and efficiently work with entire systems. User Interface software is not only relatively large and complex, but also difficult to design, to implement, and to modify. Over the years, a number of User Interface Software Tools (see box #1 in appendix), have been created to aid developers in building interfaces. The trend in these tools is toward the use of ever more complex interface primitives. We have been moving from having to program the behavior of a slide bar to be able to create complete interfaces by picking self-contained interface elements from a palette, and arranging them into a desired layout.

Unfortunately, current user interface tools continue to fail in one important aspect. Whereas there is wide consensus that the usability and acceptance of an interface are much improved whenever a *user-centered* design methodology is applied, these tools offer instead a *developer-centered* environment for interface design. Therefore, in these environments there is ample support for using and managing widgets, organizing and arranging layouts, and testing prototype interfaces. However, there is little or no support to answer key questions in any user-centered design, such as how are the widgets in a given dialog box used to accomplish the intended user tasks. Those questions can be answered only in the designer's head or via loosely connected documents.

In this paper, we introduce an emerging technology for user-centered design called *model-based interface development* (see box #2 in appendix). The basic idea of the model-based approach is that the development of a user interface can be supported in a comprehensive manner by representing all relevant aspects of a design using declarative interface models. This paradigm offers a number of key benefits:

- *User-centered development cycle.* Developers are able to design interfaces in a structured manner from abstract objects such as user tasks.
- *Centralized interface design knowledge.* Abstract elements of an interface model, such as user tasks and domain objects, share a representation language with lower level objects, such as widgets and mouse clicks. Therefore, a framework is created where all types of elements in interface models can be related among themselves. This makes it easier to, for example, determine and visualize how a given widget in an interface affects the completion of a user task.
- *Design reuse.* The availability of complete interface designs in a declarative form allows the reuse, indexing, and processing of designs for new applications.
- *Client-server framework.* An interface model defines a client-server framework for user-interface software

tools. Design tools of various types, such as model editors, design critics, decision support facilities, or automated interface generators can coexist in a single environment by communicating via a server: the interface model.

- *Runtime functions.* Usability functions and other run-time user-support activities can be linked to a declarative model. This feature enables tool-based analysis and decision support for these activities.

A typical architecture of a model-based development environment is shown in Figure 1. The interface model acts as a central repository of knowledge about an interface design. Developers access and modify the interface model via tools offering a variety of functions and levels of automated support. The tools may also utilize additional knowledge bases about design guidelines and principles to operate on the interface model or to provide advice to 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.

In the rest of this paper, we will first introduce the software development cycle for model-based systems. We will illustrate the principles and use of model-based technology via an example. Then, we will survey the state-of-the-art of the technology and assess its capabilities and limitations. Finally, we will discuss the improvements that model-based technology brings to user interface design now and its potential for impact on the future of interface development.

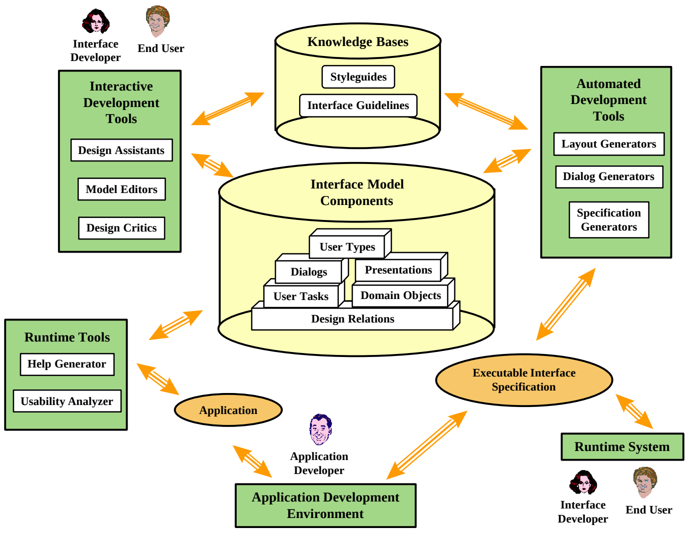


Figure 1. A generalized architecture for model-based interface development environments.

BUILDING AN INTERFACE FOR THE LOGISTICS DOMAIN

The concepts and principles of model-based interface design are best understood in the context of an example. In this section, we show how to design an interface for the logistics domain using a system called Model-Based Interface Designer (MOBI-D) [8].

MOBI-D is a precursor of an earlier model-based system called Mecano [6], which generated automatically form-based interfaces from domain models. MOBI-D introduces several innovations to the model-based field: (1) a metalevel modeling language that defines the components, structure, and relations of interface models, (2) the identification of design relations as an explicit component of an interface model, which permits a formal definition of an interface design in model-based systems (see box #2), and (3) a philosophy of interface design based on supporting the decision making of developers and end users, as opposed to one of automatic generation of interfaces prevalent in most previous model-based systems, including Mecano.

The MOBI-D Development Cycle

Figure 2 shows the MOBI-D development cycle. All the processes are fully interactive and may involve user participation. The cycle is iterative in nature. Interface design begins with the elicitation of the user tasks. An interactive MOBI-D tool allows the user to enter a textual description of the task. The tool elicits key terms such as objects and actions and guides the user in editing and refining the terms into a structured user task description. Next, The developer takes this description and builds user-task and domain models with MOBI-D model editing tools. The models are then integrated so that domain objects are related to the user tasks for which they are relevant.

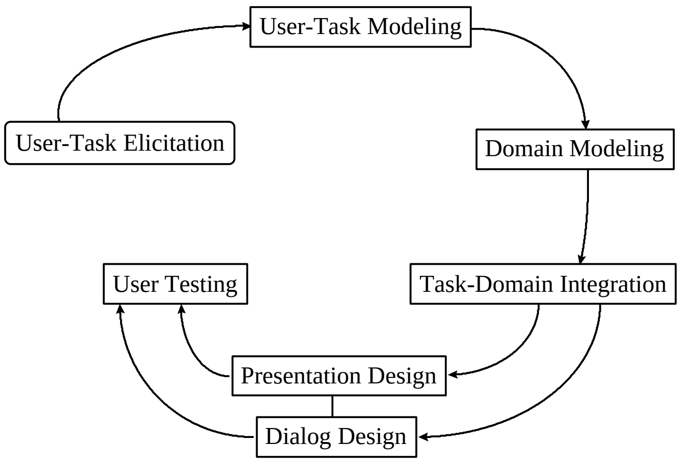


Figure 2. The interactive, user-centered development cycle in MOBI-D.

The decision support mechanisms in MOBI-D use the user-task and domain models to make recommendations for presentation and interaction techniques. These recommendations are displayed to the developer to guide the design and 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 pre-configured 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 suggested by the model. The developer could select the slider, position it in the dialog window, resize it and edit its label, color, etc. The resulting interface is tested by the end-user.

The Sample Logistics Interface

We will use MOBI-D to design an interface that allows a requisitions officer to select the best possible supply sources for a given supply. The officer can view a list of available suppliers and rank them according to a set of criteria, such as how much surplus they have and how quickly they can deliver. At any time, the officer can select a supply source and evaluate the transportation risks from source to destination. Once a supplier has been established, the order can be placed either via e-mail or by phone. This example is a subset of a much more complex interface for this domain that is being developed using MOBI-D.

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 (see appendix) shows two steps in the elicitation phase. The end user first describes the task informally not worrying how it may eventually be converted into a computer model. Then, the end user identifies key “actions” and “things” relevant to the task, as shown in (a). Finally, in (b) with the help of the developer, the end user 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 can be used directly by the developer 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 for the interface design. An skeleton of these models is read directly from the task outline and is then refined by setting properties for each subtask and domain object. Further consultation with the end user is likely during this phase. The user-task and domain models are the foundation of an interface design. The decision support tools in MOBI-D provide recommendations for presentation and dialog design based on the structure and properties of these models.

Figure 4 (a) and (b) (see appendix) 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 while the bottom pane shows the properties for the selected object. The top right pane lists prototype objects available for use in the current model via a drag-and-drop operation. An object from the current model can in turn be made a prototype by dragging it into the top right pane. Because the prototype objects are organized as user-task or domain models, they can be saved separately from the current model and then reused in a future design.

Figure 4 (c) shows the integration of the user-task and domain models that the developer has completed. When a domain element is assigned to a specific user task, a *design relation* is created. The set of all design relations in an interface form an *interface design* in MOBI-D. The user-task to domain element relations are later expanded to include presentation and dialog elements. Because these relations are stored explicitly in MOBI-D interface models, a framework for the rational visualization and reuse of interface designs is created.

Presentation and Dialog Design

The design of the presentation and dialog of an interface in MOBI-D are parallel activities. Decision support tools examine the user-task and domain models, along with perhaps a set of interface guidelines, and make recommendations for the widgets or interaction elements (interactors) that should be used to complete each of the subtasks in the user-task model. The developer, however, can override any of the recommendations made and choose different elements.

Figure 5 (see appendix) shows a snapshot of the process of dialog and presentation design. 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 layout selected interactors. MOBI-D steps the designer through each of the subtasks in the user-task model and ranks available interactors according to guidelines and to the type of data that must be displayed by each interactor. The developer is free to choose any of the available interactors. By following this process, MOBI-D makes sure that each of the user tasks defined in the user-task model is appropriately displayed and completed in the resulting interface.

Because of the clear connection between the presentation and dialog design and the user-task and domain models, MOBI-D facilitates obtaining input and advice from the end user during this phase. The end user can better visualize how each interactor relates to the task outline built at the beginning of the development cycle, therefore allowing informed decisions and a better understanding of the impact of each interface element on the overall task. When the interface design is completed, as shown in Figure 6 (see appendix), the end user can test it and again provide feedback in a way that is related to the constructed interface model.

STATE-OF-THE-ART IN MODEL-BASED DEVELOPMENT

Not surprisingly, the evolution of model-based interface development environments has closely paralleled that of interface models. We have moved from early abstract models and application data models, to partial explicit models, to current comprehensive ones. Throughout the same period, we have seen an increased availability of ever more complex interface primitives, from elementary toolkit library components (e.g., a slide bar), to basic textual or graphical editors, to prepackaged elements such as ActiveX controls that can be used to construct interfaces piecewise. The confluence of both factors: comprehensive interface models and complex primitives gives model-based development a solid foundation for its paradigm. The technology is by nature limited in applicability if it does not have rich interface models or if, because there are no suitable primitives available, it must model at a very low level of detail (i.e., almost to the level of regular programming).

Early interface models were in essence abstract and served the purpose of defining the components of and interface and their functionality [5]. With few exceptions, these models have no computational equivalent and were utilized to guide the design of interfaces and of interface development environments. One example of a system based on an abstract interface model is L-CID [5], which implements a model of an intelligent interface in a blackboard architecture. L-CID was used to rapidly prototype machine-learning based interaction techniques for user interfaces.

Historically, the first type of explicit model that appears in model-based interface development is the *data model*. This model is borrowed directly from the data structures defined by the application. It proved to be useful in generating the widgets of an interface by matching data types with widget types. Systems such as UIDE [1] and Don [1] employed data models and applied some type of layout algorithm to produce complete static interface layouts. However, no specification of the behavior of the interface could be obtained from the data models. Current model-based environments typically use data definitions as a subcomponent of their interface model.

Advances in software engineering allowed model-based systems to move beyond simple data models. Thus, systems started to utilize entity-relationship data models (Genius [2]), enhanced data models (Trident [9]), or object-oriented data models (FUSE [3]). Eventually, these models lead to fully declarative domain models (Mecano [6]) that could express effectively the relationships among the objects in a given domain. As a consequence, it was possible to generate automatically partial specifications of the dynamic behavior of an interface along with its static layout.

Though elaborate, domain models do not describe semantic functions belonging to the application functional core that are associated to objects in a domain. For this purpose, an *application model* is introduced in various forms by some model-based systems, including among others UIDE [1], Trident [12], and Humanoid [10]. The purpose of these models is to facilitate the declaration of interface behavior. For example the UIDE application model consists of *application actions*, *interface actions*, and *interaction techniques*. Parameters, pre-conditions, and post-conditions are assigned to each action and then used to control the user interface at runtime.

Application and domain models are considered partial interface models because they are limited in scope and do not cover all the relevant aspects of an interface design. Naturally, other partial interface models have emerged through the years. Most notably, these include user-task models, dialog models, and presentation models. Of these, the most crucial in supporting a user-centered design philosophy is the user-task model. This model describes the tasks that an end user performs and for which interaction capabilities must be designed. It 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 [15], FUSE [3], Tadeus [9], and Trident [12] all embed various forms of task models. The user-task model represents a significant advance in the model-based development field. 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.

A dialog *model* is used to describe the human—computer conversation. It specifies when the end user can invoke functions through various triggering mechanisms (e.g. push button, commands) and interaction media (e.g. voice input), 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 been investigated and showed evidence of success: dialog nets (Genius [2], Tadeus [9]), attributed grammars (Boss [3]), state-transition diagrams (Trident [12]), dialog templates (Humanoid [10]). However, no consensus of an ultimate dialog modeling technique has emerged.

The *presentation model* specifies how interaction objects (or widgets) appear in the different dialog states. This model generally consists of a hierarchical decomposition of the possible screen displays into groups of interaction objects. By definition, presentation and dialog models are closely inter-related. This is why some model-based development environments consider them together: ITS [14] 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.

Other partial models that have been defined but that have been seldom used in practice include *platform models*, *user models*, and *workplace models*. In general, these representations define the characteristics associated with each of their target elements. Although recognized as important, little software support has been developed for interface design based on these models. In practice, many of the characteristics of these representations are subsumed into the presentation and dialog models of some systems.

The wealth of partial interface models has eventually led to current efforts to define comprehensive interface models. This is the goal of systems such as MOBI-D [8] and Mastermind [11]. In particular, MOBI-D has created a metalevel modeling language to define the structure and organization of interface models, and has defined a specific component, called a *design model*, to specify in a declarative manner the relations among all the various elements of an interface model [7]. These comprehensive interface models are the foundation to build effective model-based interface development systems that support a robust development cycle and a broad variety of interface designs.

THE IMPACT AND FUTURE OF MODEL-BASED SYSTEMS

There are a number of areas where model-based interface development can significantly impact the way interfaces, and consequently applications, are built. We examine here some of those areas:

User-centered design. Model-based systems such as MOBI-D establish a clear channel of communication between the end user, the interface developer, and between them and the interface design. Most importantly, this communication is grounded on a computational model with defined operators and software support. It is precisely such a context that enables user-centered design within a rational, well-connected methodology. In many instances, user-centered design fails because of poorly understood or weak connections between the task analysis phase and the implementation phase. A model-based environment can engage the end user in the development process always keeping a rational view of how design decisions affect user tasks.

Usability and user studies. The potential of the model-based paradigm to provide support beyond the design and development phases must not be underestimated. One of the areas where this is particularly true is that of usability and user studies. The design of experiments, collection of data, and analysis of results can all be supported and integrated via interface models. Thus, the actions of an actual user during an interaction session can be mapped precisely to a user-task model. Deficiencies in usability can be corrected 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 offered software support.

Model-based environments as advice tools. Recent and future model-based environments promote the idea of *supporting* interface design as opposed to *automating* it. They should be thought of as a computer-aided design tools, i.e. advice tools that provide assistance to designers of interfaces: for each design option, the system proposes accurate and reliable values by relying on design knowledge. The recommendations, however, do not limit the flexibility of the designer but rather organize the decision making process of that designer. 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.

Model-based technology and the Internet. As we remarked earlier, one of the keys to the success of model-based technology is the availability of interface primitives that can be treated by the systems during interface design as encapsulated objects with predefined functionality. This feature makes model-based development environments a particularly 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.

CONCLUSIONS

The model-based technology and the development cycle established by MOBI-D address two current key problem areas in user-interface development: (1) The need for user-centered design environments, and (2) the lack of comprehensive software systems that support all major stages of development. Most of the shortcomings of current user-interface software tools can be traced to one or both of those problems. In contrast, MOBI-D implements the model-based paradigm in a manner that defines a clear life cycle for user-interface development where user-centered design and end-user participation are central elements. As we have seen, MOBI-D is the most recent step in the evolution of model-based systems. We expect that this technology will continue to evolve and that it will play an important role in the future of interface and application development.

ACKNOWLEDGMENTS

The work on MOBI-D is supported by the Defense Advanced Research Projects Agency (DARPA) under contract N66001-96-C-8525. David Maulsby has contributed significantly to the design of the MOBI-D tools. Our thanks to Eric Cheng, Kjetil Larsen, Justin Min, and Chung-Man Tam for their work on the development of MOBI-D.

REFERENCES

1. J.D. Foley, History, Results and Bibliography of the User Interface Design Environment (UIDE), an Early Model-based Systems for User Interface Design and Implementation, *Proc. of 1st Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'94*, F. Paternó, ed., Focus on Computer Graphics Series, Springer-Verlag, Berlin, 1995, pp. 3-14. http://www.info.fundp.ac.be/~jvd/dsvis/dsvis94.html
2. C. Janssen, A. Weisbecker, and J. Ziegler, Generating User Interfaces from Data Models and Dialogue Net Specifications, *Proc. of the ACM Conference on Human Factors in Computing Systems INTERCHI'93*, ACM Press, New York, 1993, pp. 418-423.
3. F. Lonczewski and S. Schreiber, The FUSE-System: an Integrated User Interface Design Environment, in [16], pp. 37-56.
4. B.A. Myers, User Interface Software Tools, *ACM Trans. Computer-Human Interaction*, Vol.2, No.1, March 1995, pp. 64-103.
5. A. Puerta, The Study of Models of Intelligent Interfaces, *Proc. of the 1993 International Workshop on Intelligent User Interfaces*, Orlando, Florida, Jan 4-7,1993, ACM Press, pp. 71-78.
6. A. Puerta, Model-Based Automated Generation of User Interfaces, *Proc. of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, July 31 - August 4, 1994, MIT Press, pp. 471-477.
7. A. Puerta, The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development, in [16], pp. 19-35.
8. A. Puerta, Management of Interface Design Knowledge with MOBI-D, *Proc. of the 1997 International Conference on Intelligent User Interfaces*, Orlando, Florida, Jan 6-9,1997, ACM Press, (in press).
9. E. Schlungbaum and T. Elwert, Automatic User Interface Generation from Declarative Models, in [16], pp. 3-18.
10. P. Szekely, P. Luo, and R. Neches, Beyond Interface Builders: Model-Based Interface Tools, *Proc. of the ACM Conference on Human Factors in Computing Systems INTERCHI'93*, ACM Press, New York, 1993, pp. 383-390.
11. P. Szekely, P. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Salcher, Declarative Interface Models For User Interface Construction Tools: The MASTERMIND Approach, in *Engineering for Human-Computer Interaction*, Proceedings of EHCI'95, L. Bass and C. Unger, eds., Chapman & Hall, London, 1995, pp. 120-150.
12. J. Vanderdonckt and F. Bodart, Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection, *Proc. of the ACM Conference on Human Factors in Computing Systems INTERCHI'93*, ACM Press, New York, 1993, pp. 424-429.
13. J. Vanderdonckt, Computer-Aided Design of User Interfaces, *Proc. of CADUI'96*, Presses Universitaires de Namur, Namur, 1996. http://www.info.fundp.ac.be/~jvd/dsvis/ cadui96.html.
14. C. Wiecha, W. Bennett, S. Boies, J. Gould, and S. Green, S., ITS: A Tool for Rapidly Developing Interactive Applications, *ACM Trans. on Information Systems*, Vol. 8, No. 3, July 1990, pp. 204-236.
15. S. Wilson, P. Johnson, Bridging the Generation Gap: From Work Tasks to User Interface Designs, in [16], pp. 77-94.

APPENDIX

BREAK-OUT BOX #1: USER INTERFACE SOFTWARE TOOLS

User interface software tools exist in a number of different forms, but can be classified generally into two main groups: Toolkits and higher-level development tools.

Toolkits are libraries of primitive interface components, such as menus, buttons, and scroll bars that can be called by application programs. Toolkits tend to be tedious in use because they may contain hundreds of procedures. It is often not clear how to use the procedures to create a desired user interface. The toolkits are designed to be used by programmers.

Higher-level development tools, such as user-interface management systems and interface builders, are built on top of toolkits. They are primarily developer-oriented although non-programmers may be able to utilized them in a limited way. User-interface management systems normally provide a special purpose language to specify the user interface syntax (i.e. the sequences of input and output actions), to make the user interface software production process easier, and to allow non-programmers to participate in the creation of a user interface. The systems are not effective in capturing all the relevant design knowledge of an interface. With an interface builder, a developer specifies the components and layout of a user interface through direct manipulation using a palette of primitive interface elements. Interface builders make it easy to specify the static layout of an interface. However, the specification of user interface behavior is not well supported.

Reference: B.A. Myers: *User Interface Software Tools*. ACM Transactions on Computer-Human Interaction, Vol.2, No.1, March 1995, pp. 64-103.

BREAK-OUT BOX #2: GLOSSARY OF KEY TERMS IN MODEL-BASED DEVELOPMENT

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 submodels (e.g., a user-task model, or a presentation 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.

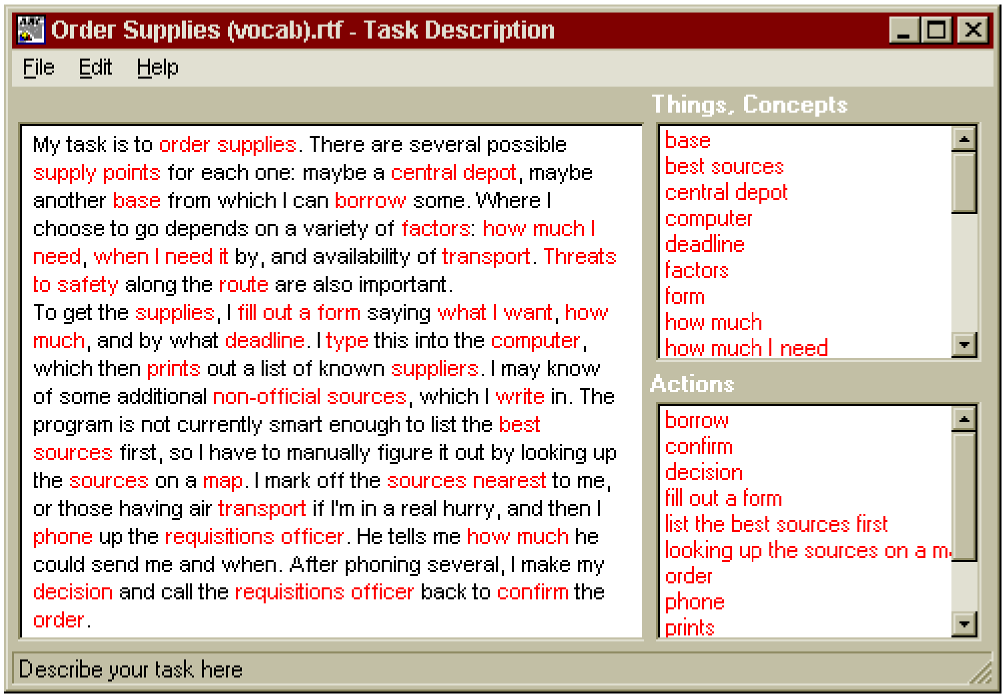
Interface design
A set of relationships among interface objects in an interface. It answers the question of how, for example, a given widget relates to a dialog structure, to a presentation scheme, to a domain element, and to a user task.

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.

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 (e.g., a model-editor), or automated tools (e.g., a layout generator).

Runtime tools
The set of software tools that use an interface model to support end user activities, such as generation of animated help or collection and analysis of usability data.

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


Order Supplies (vocab).rtf - Task Description

File Edit Help

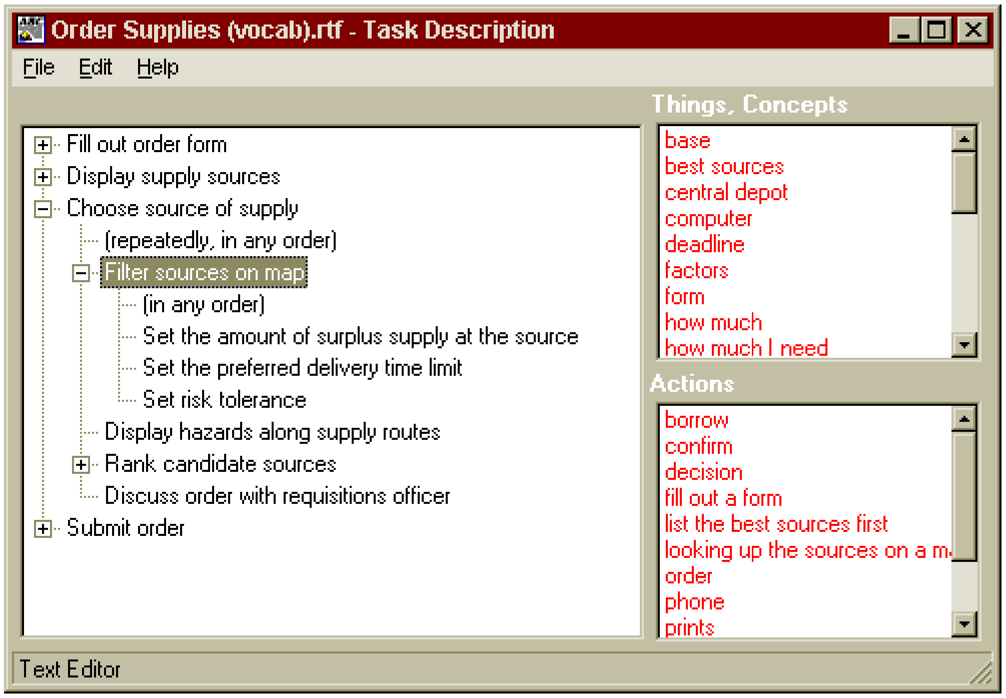
My task is to order supplies. There are several possible supply points for each one: maybe a central depot, maybe another base from which I can borrow some. Where I choose to go depends on a variety of factors: how much I need, when I need it by, and availability of transport. Threats to safety along the route are also important.
To get the supplies, I fill out a form saying what I want, how much, and by what deadline. I type this into the computer, which then prints out a list of known suppliers. I may know of some additional non-official sources, which I write in. The program is not currently smart enough to list the best sources first, so I have to manually figure it out by looking up the sources on a map. I mark off the sources nearest to me, or those having air transport if I'm in a real hurry, and then I phone up the requisitions officer. He tells me how much he could send me and when. After phoning several, I make my decision and call the requisitions officer back to confirm the order.

Things, Concepts
base
best sources
central depot
computer
deadline
factors
form
how much
how much I need

Actions
borrow
confirm
decision
fill out a form
list the best sources first
looking up the sources on a m
order
phone
prints

Describe your task here

(a)


Order Supplies (vocab).rtf - Task Description

File Edit Help

- Fill out order form
- Display supply sources
- Choose source of supply
  - (repeatedly, in any order)
  - Filter sources on map
    - (in any order)
    - Set the amount of surplus supply at the source
    - Set the preferred delivery time limit
    - Set risk tolerance
  - Display hazards along supply routes
  - Rank candidate sources
  - Discuss order with requisitions officer
- Submit order

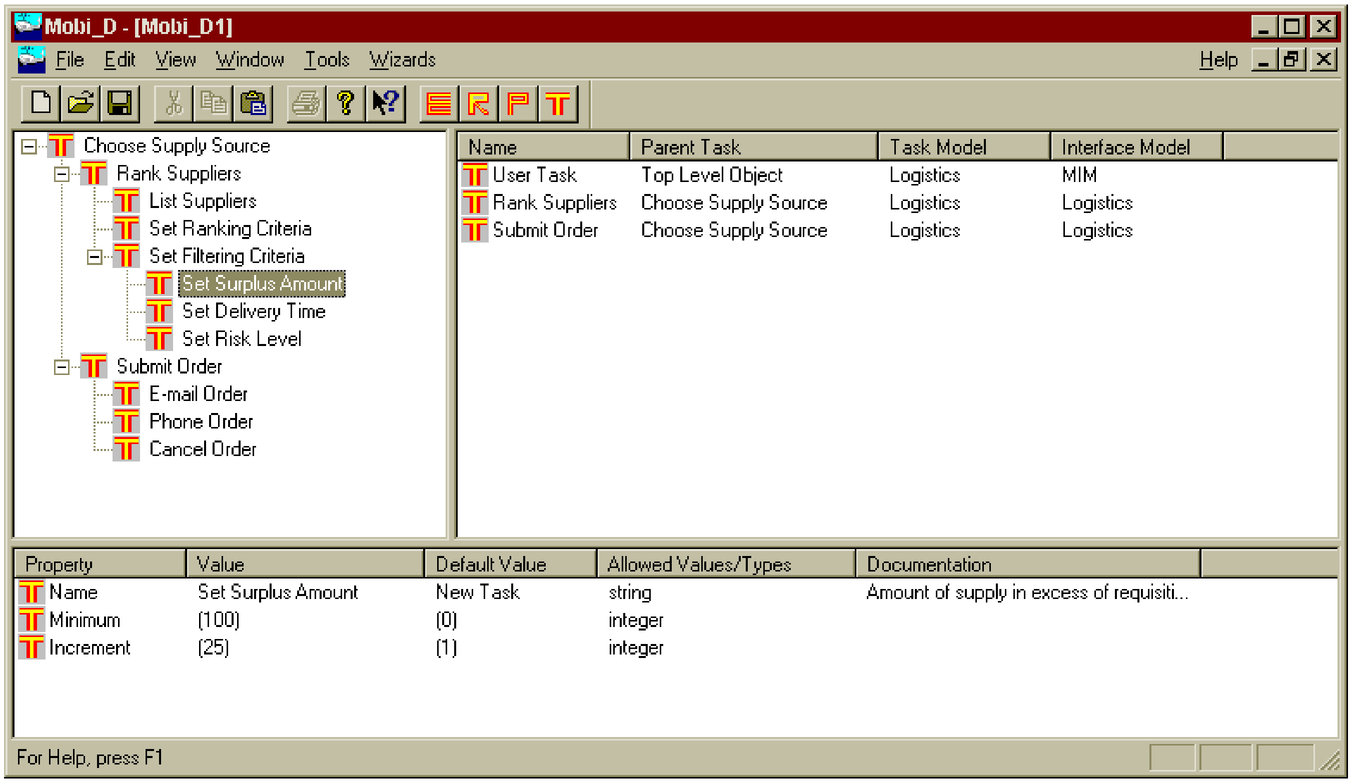
Things, Concepts
base
best sources
central depot
computer
deadline
factors
form
how much
how much I need

Actions
borrow
confirm
decision
fill out a form
list the best sources first
looking up the sources on a m
order
phone
prints

Text Editor

(b)

Figure 3. Two steps in the user-task elicitation stage of MOBI-D.

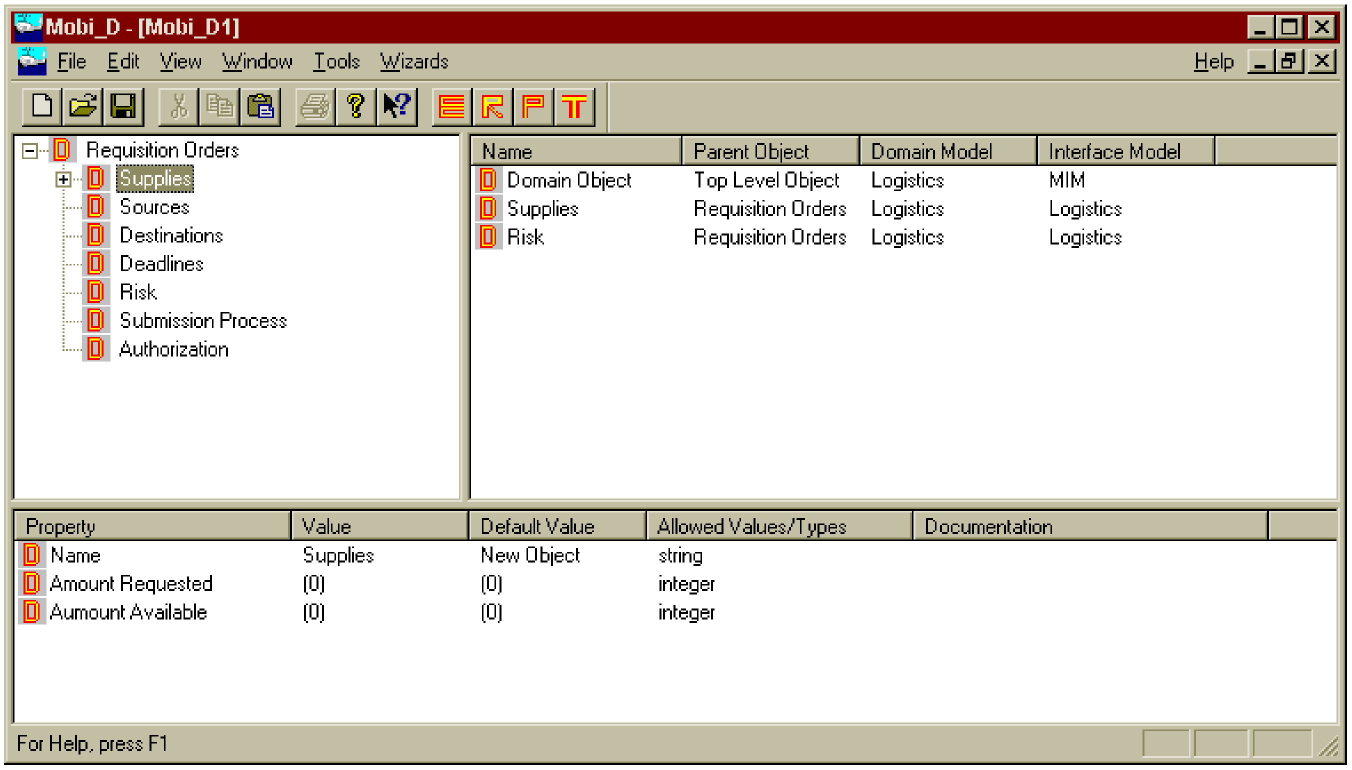


- Choose Supply Source
  - Rank Suppliers
    - List Suppliers
    - Set Ranking Criteria
    - Set Filtering Criteria
      - Set Surplus Amount
      - Set Delivery Time
      - Set Risk Level
  - Submit Order
    - E-mail Order
    - Phone Order
    - Cancel Order

Name	Parent Task	Task Model	Interface Model
User Task	Top Level Object	Logistics	MIM
Rank Suppliers	Choose Supply Source	Logistics	Logistics
Submit Order	Choose Supply Source	Logistics	Logistics

Property	Value	Default Value	Allowed Values/Types	Documentation
Name	Set Surplus Amount	New Task	string	Amount of supply in excess of requisiti...
Minimum	(100)	(0)	integer	
Increment	(25)	(1)	integer	

(a)



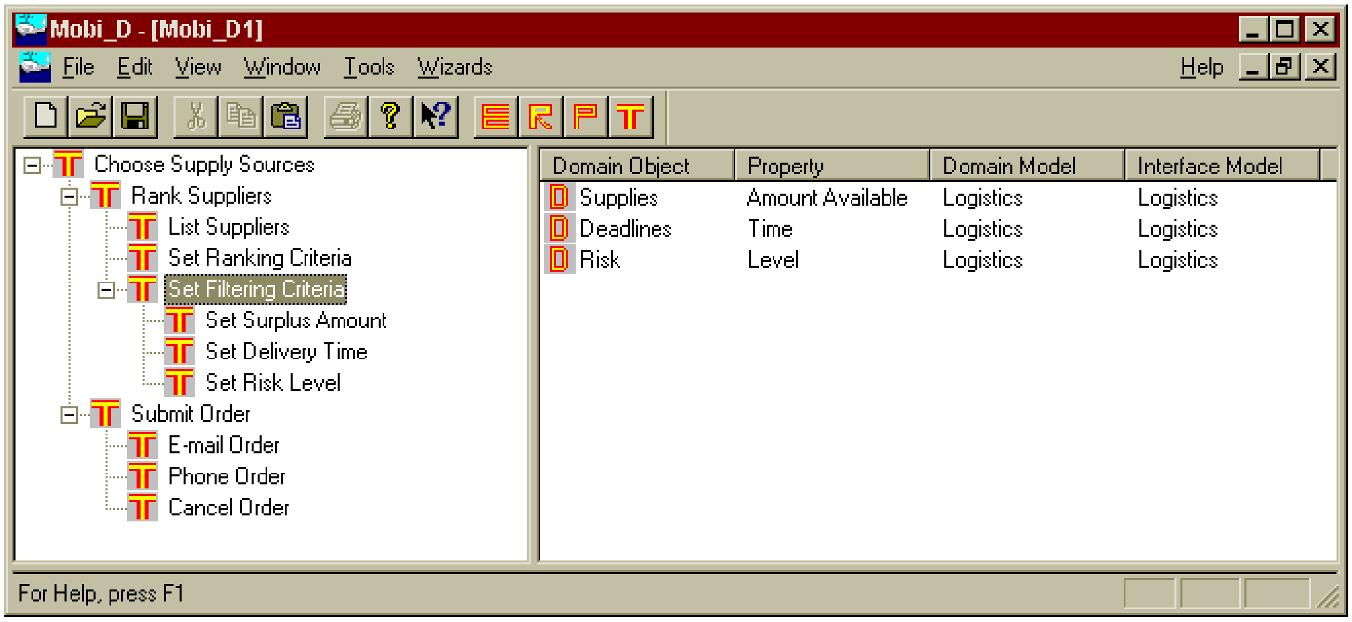
- Requisition Orders
  - Supplies
  - Sources
  - Destinations
  - Deadlines
  - Risk
  - Submission Process
  - Authorization

Name	Parent Object	Domain Model	Interface Model
Domain Object	Top Level Object	Logistics	MIM
Supplies	Requisition Orders	Logistics	Logistics
Risk	Requisition Orders	Logistics	Logistics

Property	Value	Default Value	Allowed Values/Types	Documentation
Name	Supplies	New Object	string	
Amount Requested	(0)	(0)	integer	
Aumount Available	(0)	(0)	integer	

(b)

Figure 4 (a) and (b). User-task and domain model editors in MOBI-D.



- Choose Supply Sources
  - Rank Suppliers
    - List Suppliers
    - Set Ranking Criteria
    - Set Filtering Criteria
      - Set Surplus Amount
      - Set Delivery Time
      - Set Risk Level
  - Submit Order
    - E-mail Order
    - Phone Order
    - Cancel Order

Domain Object	Property	Domain Model	Interface Model
Supplies	Amount Available	Logistics	Logistics
Deadlines	Time	Logistics	Logistics
Risk	Level	Logistics	Logistics

(c)

Figure 4 (c). Integration of user-task and domain models

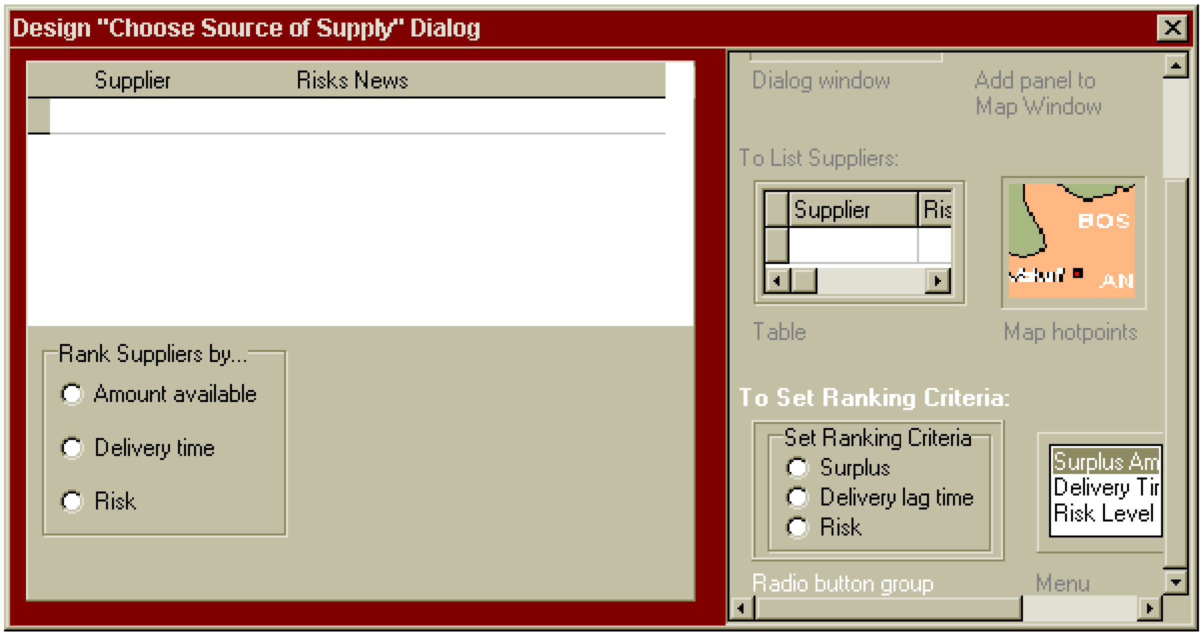
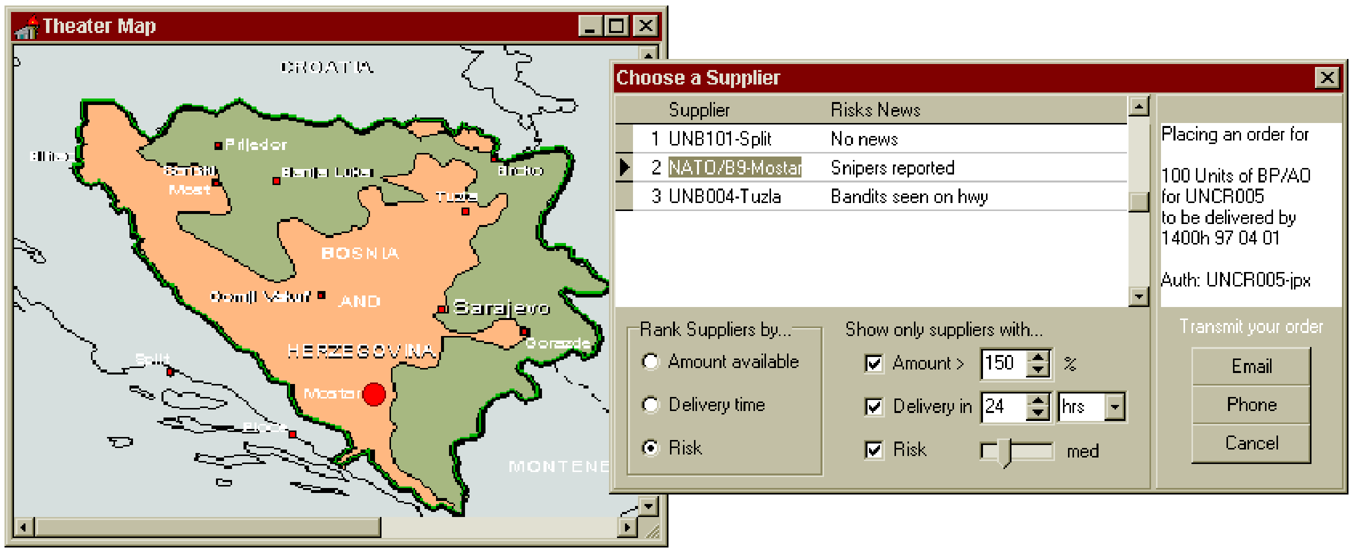


Figure 5. Designing the presentation and dialog with MOBI-D.



	Supplier	Risks News
1	UNB101-Split	No news
2	NATO/B9-Mostar	Snipers reported
3	UNB004-Tuzla	Bandits seen on hwy

Rank Suppliers by... [✓ Risk]

Show only suppliers with...
- ✓ Amount > 150 %
- ✓ Delivery in 24 hrs
- ✓ Risk med

Placing an order for

100 Units of BP/AO
for UNCR005
to be delivered by
1400h 97 04 01

Auth: UNCR005-jpx

Figure 6. The interface is now ready for user testing.