MOBILE: User-Centered Interface Building

Angel R. Puerta, Eric Cheng, Tunhow Ou, Justin Min,
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
Interface builders are popular tools for designing and developing graphical user interfaces. These tools, however, are engineering-centered; they operate mainly on windows and widgets. A typical interface builder does not offer any specific support for user-centered interface design, a methodology recognized as critical for effective user interface design. We present MOBILE (Model-Based Interface Layout Editor) an interface building tool that fully supports user-centered design and that guides the interface building process by using user-task models and a knowledge base of interface design guidelines. The approach in MOBILE has the important added benefit of being useful in both top-down and bottom-up interface design strategies.

Keywords
Model-based interface development, task models, interface builders, user-centered interface design, user interface development tools.

INTRODUCTION
For a good number of years, interface-building tools have gained wide acceptance among developers of graphical user interfaces [3]. Interface builders allow developers to layout and organize, via direct manipulation, the various elements of a graphical user interface (GUI). Typically, these tools include code generators that produce the basic hooks for application developers to write the code to communicate with the user interface. All major commercial software-development environments currently available include interface-building tools.

Although efficient at what they do, interface builders restrict their scope to manipulation of those elements that make up a GUI, such as windows and widgets. They support, in essence, an engineering process. An interface developer working with an interface builder occupies his or her thoughts with widget selection, layout, and organizational issues. Any connection between the operations made through an interface builder and the requirements of the target users and their tasks must be maintained in the head of the developer without assistance from the interface-building tool.

Separately, the user-interface community has come to accept that one of the best methodologies for user-interface construction is that of *user-centered design* [4]. The basis of this methodology is straightforward: The design of a user interface should be guided principally by the nature of the task that the user needs to accomplish. This differs from so-called engineering-centered approaches where interface design decisions are made according to the requirements of the application being built. The benefits of user-centered design have been clearly demonstrated over the years [4].

It is therefore curious that the clearly effective graphical interface builders do not support the similarly effective user-centered approach. This opens the question of how could interface builders be augmented, enhanced, or modified in order to enable a user-centered approach but without changing the operations in such a way that the original benefits of the tools disappear.

OUR SOLUTION
The approach taken by our group incorporates elements of user-centered design and of model-based interface development into the functionality of an interface builder. From user-centered design we take the idea of building user-task representations as a guide for interface development. From model-based interface development [8] we take the ability to create, edit and refine user-task models. These models are computational units that can be exploited by an interface builder. Finally, from interface builders we take their basic functionality and try to augment it in very specific ways (using user-task models) to enable a user-centered process.

The result is a tool called MOBILE (Model-Based Interface Layout Editor), which enables user-centered interface building. MOBILE allows developers to

interactively build user interfaces according to a user-task model. The tool also provides decision-support guidance thanks to knowledge base of interface design guidelines. Users of MOBILE can benefit whether they use a top-down approach (i.e., build a user-task model and then an interface), or a bottom-up one (i.e., build a user interface and construct a user-task model that goes along with it).

The rest of the paper is organized as follows. We first provide some contextual information about model-based interface development. Then, we describe MOBILE and its main functional characteristics. We illustrate the use of the tool via a sample target interface. We proceed by detailing the decision-support capabilities of MOBILE and by describing its use in a bottom-up approach. We conclude by relating our evaluation experiences, the work related to our approach, and the possible directions of future research.

MOBI-D

Model-based interface development [8] is a technology that embraces the idea of designing and developing user interfaces by creating *interface models*. An interface model is a computational representation of all the relevant aspects of a user interface. The components of an interface model include submodels such as user-task models, domain models, user models, presentation models and dialog models. Model-based interface development systems are suites of software tools that allow developers to create and edit interface models. Many model-based systems aim at generating significant parts of a user interface given a partial interface model. Some others aim at interactively guiding developers in building user interfaces using interface models [8].

Over the past three years, our group has been developing MOBI-D (Model Based Interface Designer) [8]. MOBI-D is a model-based interface development environment that enables designers and developers to interactively create user interfaces by defining interface models. The environment integrates a variety of tools including model-editing tools, user-task elicitation tools, and the interface building tool presented here.

A full description of the model-based interface technology and development methodology supported by MOBI-D has been presented elsewhere [8] and it is beyond the scope of this paper. Some of the other individual tools integrated into MOBI-D have also been described in previous publications [5, 7, 9]. For our purposes, however, we simply need to note that a component of the interface models constructed in MOBI-D is a user-task model. This component is the essential element for the interface-building tool presented in this paper.

User-Task Models

A user-task model in MOBI-D is a hierarchical representation of a user's task. The model decomposes the user task into subtasks arranged in a tree-like structure. Attributes can be specified for any task as well as procedural information (e.g., whether certain sub tasks must be executed in sequence). Conditions that affect the execution of a task/subtask can also be specified in the model. Domain objects (and their attributes) involved in the completion of a task can also be defined and associated with any task. In general, a user-task model is less complex than a workflow diagram and it can retain a certain informal level to it without losing its usefulness. In MOBI-D, user-task models are elicited from domain experts and then refined by interface developers [9]. Once created, it is available to any of the other tools in the environment.

SAMPLE INTERFACE

For illustration purposes of some of the shortcomings of conventional interface builders, let us consider the partial interface shown in Figures 1 through 3, which has been designed using the MOBI-D tools. These figures show screen snapshots from a military logistics application. This application allows users to perform typical tasks associated with requesting and monitoring supplies in a theater of operations. These tasks include among others: (1) creating and modifying plans for requisitions of materials, (2) reviewing potential suppliers for location, available stocks, and delivery times, (3) requesting supplies and tracking shipments, and reviewing all current stocks of materials

The application supports users of different ranks. The dialog and presentation should adapt to the rank of the user and to the specific task that the user must perform. In the screen snapshots we can observe the following situations:

Figure 1 is the initial screen (after login) for a user of rank Major. Typically, a user of this application needs to see the *authorized stock levels* (ASLs) for the current operation and needs access to a map of the region. The Major can inspect the ASLs via the 3-D viewer shown on the left. She can change the data in the viewer to that of a different location by clicking on the particular location on the map shown on the right. Each row of bars in the 3-D viewer (lengthwise) corresponds to a different class of materials (e.g., subsistence items, ammunition). The user can quickly see in this viewer if any class has a deficiency in ASLs in which case, he is authorized to modify it and does so via the push button shown above the 3-D viewer. We determined from domain experts and from the construction of a user-task model for this interface that this screen fulfills the first activity that a user must perform (overview of operation) and that the data presented was exactly (not more or less) what is needed to complete the overview.

Figure 2 shows the initial screen for a user of rank Sergeant. This user can also inspect ASLs according to

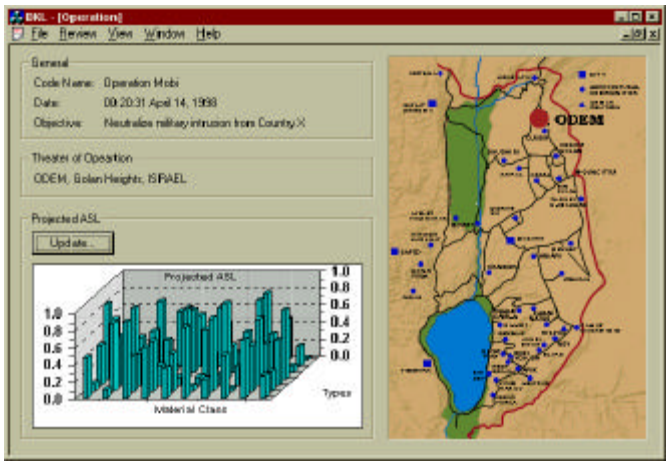

Figure 1. Initial screen in logistics application for a user of rank Major.

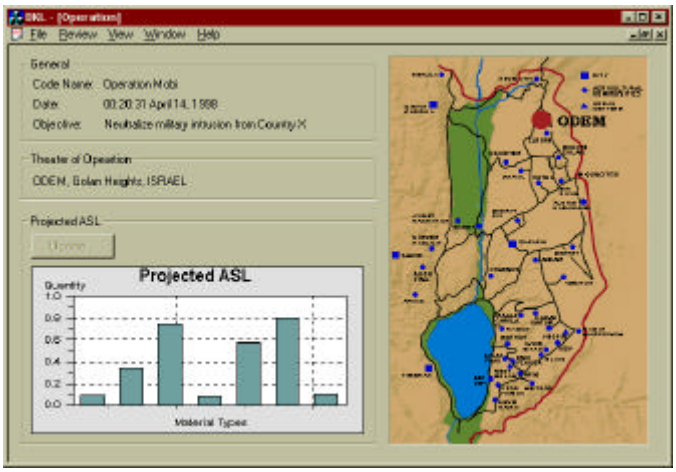

Figure 2. Initial screen in logistics application for a user of rank Sergeant.

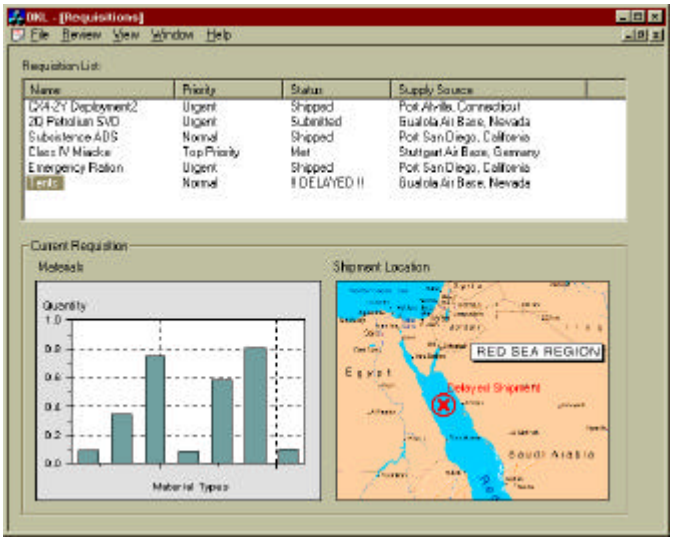

Figure 3. Shipment inspection screen for a user of rank Sergeant.

location but since her task is different to that of a Major, the dialog and presentation adapt. The central role of a Sergeant in this scenario is to carry out the requisition plans constructed by the major. The Sergeant observes when supplies are running low and orders new shipments that conform to the levels set by the Major. Because a Sergeant deals only with one specific class of materials (e.g., subsistence items only), the interface uses a 2-D viewer for the ASLs. In addition, since a Sergeant is not authorized to modify ASLs, the pushbutton for access to the ASL modification screens is disabled. The user-task information is again derived from the accompanying user-task model.

Figure 3 shows a shipment inspection screen for a user of rank Sergeant. Once more, the complete information needed to perform the inspection task is included in this screen as dictated by the user-task model. Interestingly, under certain conditions (e.g., shipment delay) this screen must be the initial screen for a Sergeant user (as opposed to that of Figure 2). In such a situation, the decision context of the Sergeant changes from one of *monitoring* (as in Figure 2) to one of *repair* (e.g., request new shipment, wait, reroute other shipment).

Clearly, a conventional interface builder can be used to layout and arrange the elements of any of the screens discussed above. However, such a tool would offer no help with managing any of the user-task requirements. Issues such as how data should be split among the screens, what widgets correspond to what type of user, and how the dialog changes according to the task and user characteristics are well beyond the support of a typical interface builder. In practice, it may be that such user-task information is kept in paper documents, or is viewable through a separate tool, or (worse) it is just in the head of the designer. The result is bound to be a number of mismatches between the designed screens and the user-task specifications. In addition, revisions of the screens or of the specifications can produce even more pronounced mismatches, or at the very least a cumbersome coordination process.

We aim for a much higher level of coordination and support for user-task specifications in the interface building process. This is the central goal of MOBILE.

MOBILE

MOBILE (Model-Based Interface Layout Editor) is an interactive software tool for user-centered interface building. Figure 4 shows the main architectural components of MOBILE. A task/presentation manager communicates with an interface model to obtain and update information related to user-task models and presentation elements of the target interface. The interface model is also used by a knowledge base of interface design guidelines to manage a palette of standard and custom widgets that interface designers use to select elements for layout.

The general philosophy in MOBILE is to guide and facilitate the interface building process. It is never to automate the creative decision facets of that process.

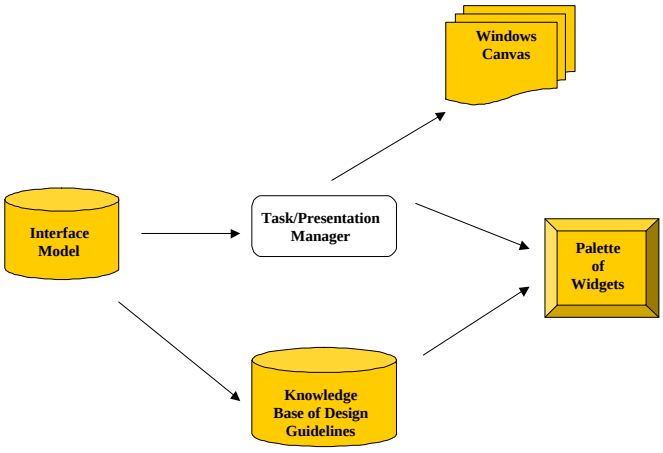

Figure 4. Architectural components of MOBILE.

Functionality

Figure 5 shows the main functional elements of MOBILE during the design of the screen shown in Figure 1. The task/presentation manager is shown to the left as a split-screen view. The palette of widgets (top right) is a toolbar where each icon represents an available widget. The canvas area (bottom right) is the drop target for selected widgets where interface elements are arranged under the direction of the interface designer.

The right-bottom pane of the task/presentation manager is the user-task model inspector. Here the interface designer can review the hierarchy of tasks and subtasks. Limited operations are possible in this pane. The designer can add a new task (by clicking on the icon just above the task inspector), delete tasks, or regroup them in a different order. More elaborate operations on the user task model (e.g., setting task attributes) require moving to a separate model-editing tool in MOBI-D. This separation is by design and was determined by our evaluation of MOBILE (see evaluation section). Immediately above the task inspector is also the end-user selector pull-down list. Changing the user selected in this list results on the task inspector being updated with the user-task model for the specific user type selected.

The left-bottom pane of the task/presentation manager is the presentation inspector. Here the interface designer can review the elements of the interface and their relationship to the user-task elements. The top elements in the presentation trees are windows (a window can be a regular window or a dialog panel). Underneath each window, as per the tree hierarchy is the task(s) assigned to that

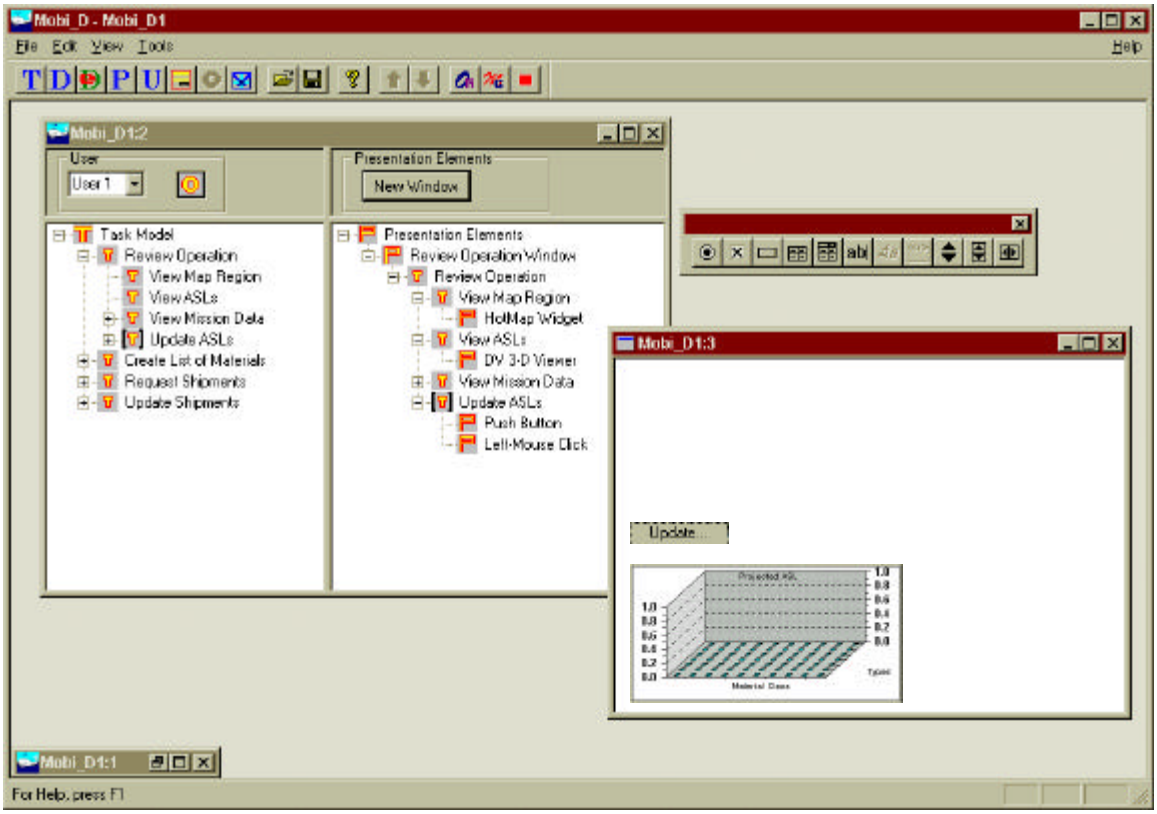

Figure 5. The main functional elements of MOBILE: The task/presentation manager on the left, the palette of widgets on the top right, and a windows canvas area.

particular window by the designer. Each task can have any number of subtasks. The leaf elements of the presentation tree are found under subtasks and include the widget that allows completion of the subtask (e.g., a push button, a 3-D viewer), and an interaction technique (e.g., a left-mouse click). Immediately above the presentation inspector there is a button to create new windows in the presentation tree.

The palette of widgets is a toolbar populated with icons each symbolizing an available widget. The widgets can vary from standard ones, such as checkboxes and text fields, to complex ones, such as the 3-D viewer shown in Figure 1. These widgets are not created and maintained by us. They are strictly third-party elements. For example, the 3-D viewer is an ActiveX control supplied by a company called DataViews [10]. In general, the MOBILE palette can access standard Windows95 widgets and any other widget wrapped as either a Java Bean or an ActiveX control. The canvas areas in MOBILE are identical to those of conventional interface builders.

A typical sequence of designer operations in MOBILE is as follows. The designer starts the tool and selects a target end user via the user selector. This updates the user-task inspector with the user-task model corresponding to the selected end user. The presentation inspector appears initially empty (except for the root element). The designer creates one or more new windows that are inserted automatically into the presentation tree. The next step is to assign tasks from the user-task model to specific windows. This is accomplished by simple drag-and-drop operations. If a dragged task contains any subtasks, these are also included in the same window as the parent task (the designer can later change this assignment).

Following task assignment, the designer selects specific windows within the presentation tree. For each window, the designer accepts or modifies the subtasks that are grouped into that window (by visual inspection). This may require merging or splitting windows if changes are desired. Ultimately, the designer reaches a satisfactory arrangement of tasks into windows. At that point, the designer then selects each leaf subtask (i.e., a subtask that has no subtasks of its own) and uses the palette of widgets to select a widget to complete that particular subtask. Additionally, the designer can select an interaction technique(s) to perform the task.

In our example, the sequence looks like this (refer to Figure 5). The designer selects *user1* (of type Major) and the corresponding user-task model is displayed. After creating a *new window*, the designer drags the task *review operation* onto the new window. This window is automatically relabeled *review operation window*. All subtasks of *review operation* are placed under the *review operation window*. For each leaf subtask (e.g., *view map*

region, *view ASLs*) the designer selects one widget from the palette to perform the operation (e.g., the 3-D viewer widget for the *view ASLs* subtask).

Note some interesting decisions from the design of the screen in Figure 1. The user-task model in the task inspector of Figure 5 shows four subtasks for the task *review operation*. The subtask update ASLs has some subtasks of its own (as revealed by the "+" sign next to its name). However, no such subtasks appear for *the update ASLs* subtask in the presentation inspector of Figure 5. The designer has noted that the update subtask is optional (this is conveyed by the somewhat different icon next to the task name in the task inspector). Because of its optional nature, the designer decides not to clutter the screen with potentially useless elements and to relegate any subtasks of *update ASLs* to a different screen. The designer simply provides a navigation button for the end user whom will use it, if necessary, to access the update functionality.

Second, the screen in Figure 2 shows that the button for *update ASLs* is disabled for a user of type Sergeant. We already discussed that users of this rank are not allowed to perform updates. In the task inspector window this results in the *update ASLs* subtask not appearing as part of the user-task model when the designer selects a user of this rank. Without any changes by the designer, the consequence would be that the screen of Figure 2 would contain no button at all for the Update of ASLs. Instead, the designer decided to add the *update ASLs* task to the user-task model of *Sergeant* and in place of *left-mouse-click* (as in Figure 5 for users of type *Major*) insert *disabled*. This decision was made for purely aesthetic reasons in this case as it avoided creating a wide blank area within the screen.

This type of close coordination between use-task requirements and interface building is the main benefit of MOBILE. Designers can evaluate at all times their interface building decisions based on the specifications of the user task. They can also effectively manage the links between the various types of users, the user-task specifications for each user, and the widgets and interaction techniques that correspond to each task and subtask. Furthermore, assignment of user-tasks into windows is a direct manipulation operation. None of these important functions are available in conventional interface builders.

DECISION SUPPORT

In addition to the basic functionality offered by MOBILE, a knowledge-based decision support system complements the assistance given by the tool to interface designers.

As we discussed earlier, a user-task model encompasses knowledge about the attributes and nature of user tasks as well as about the domain objects involved in the completion of a given task. When a designer working with MOBILE selects a subtask for which a widget must be assigned, MOBILE can exploit the user-task knowledge to assist in the assignment process.

Based on the attributes of tasks and their related domain objects, MOBILE can consult a knowledge base of interface design guidelines to determine what are the most appropriate widgets to use for a given task. The knowledge base is essentially a decision tree. The inference mechanism looks at attributes of objects, such as data types and value ranges, in order to traverse the tree to find optimal widgets. As a simple example, if a data/domain object to be accessed via the interface is of type Boolean, then the inference mechanism will recommend a checkbox as a suitable widget. The widgets identified in this manner may also be grouped into discrete categories reflecting their relative suitability (e.g., high, medium, or low).

When a designer working in MOBILE sets a preference to work in *guided mode*, the tool reflects its decision-support capabilities via the palette of widgets. As the designer selects a subtask for assignment of a widget, MOBILE disables all widgets in the palette that make no sense according to the knowledge base of interface design guidelines. In addition, MOBILE will highlight the widgets that are considered of high suitability. In this manner the attention of the user is directed towards the optimal widgets and irrelevant choices are removed from consideration.

In addition to widget assignment, MOBILE also exploits the user-task models to provide a *user-task- and domain-specific interface building experience* to the designer. For example, in conventional interface builders when the designer selects a push button from a palette of widgets and places the widget on a window canvas, the widget appears with either a generic label (e.g., "button1") or no label at all. In MOBILE, every widget assigned appears already tailored to the specific task and domains. In the case of Figure 1, the button for updating ASLs first appears on the window with that label as the information is carried directly from the user-task model. This capability serves to further solidify the user-centered design experience for the user of MOBILE.

BOTTOM-UP INTERFACE DESIGN STRATEGIES

The use of MOBILE described so far follows a strict top-down approach. First a user-task model must be built, then MOBILE can be used to lay out a corresponding interface. It can be easily argued that this limits the freedom of interface designers. Some designers like to immediately jump into an interface builder and informally construct possible designs for a user interface. Having to work out a user model beforehand may be an undesirable burden for these designers.

Fortunately, MOBILE can be used by this type of designer and still potentially provide some of the user-centered

benefits of the tool. A bottom-up approach with MOBILE would entail the same kind of free-form interface layout that is available with a conventional interface builder. However, once the designer starts settling with a particular set of layouts, the designer can *annotate* each window (and widget) with a newly created user task and then can arrange the tasks into a skeleton user-task model. In this mode, MOBILE acts as a *design rationale* tool. The initial user-task model can always be refined into a complete one that would be useful in any revision and update of the interface.

EVALUATION

The implementation of MOBILE shown here has evolved through several evaluations that also included an early mock-up and two preceding prototypes. Along the way, we learned what are the functions that designers really want in a tool like MOBILE, and how the on-screen items should be arranged for better efficiency in the interaction.

Our initial mock-up did not include a task/presentation manager. Instead it counted on the existence of a user-task model-editing tool in MOBI-D. MOBILE simply provided canvas areas for windows, each with an attached palette of widgets populated specifically for that window and its associated task. Users were quick to point out that it was cumbersome to continually switch from MOBILE to the user-task model editor. Furthermore, the model editor included lots of functions that were not relevant at interface-building time. The one-palette-per-screen approach also seemed to consume too much screen real estate.

Our first prototype was entirely task-based (i.e., a user-task inspector but no presentation inspector). The user-task inspector included only the functionality for editing user-task models that users felt was relevant (deleting, adding, and regrouping tasks). A single palette was attached to the inspector. The palette changed its widgets according to which task was selected in the user-task model inspector. Designers remarked that it was important for them to be able to see the organization of the presentation elements (this could be done but only by switching to another view within MOBI-D).

The second prototype included a task/presentation manager similar to the current one, and a dynamic palette that changed its widgets with each task/subtask selection in the manager. The main difficulty in this version was that designers did not want the palette changing continuously. This forced them to visually inspect the palette for every task to see what widgets appeared and in what order.

In the current prototype, we fixed the elements of the palette of widgets. Their location on the palette is always the same. We simply change their enable/disable state and highlight widget icons if necessary, as detailed on the earlier section on decision support. We also don't make any changes to the palette if the user is selecting tasks/subtasks in the user-task inspector (as we did in previous versions). We only modify the palette when the user selects a subtask in the presentation inspector. In the earlier version, it caused confusion for users to be inspecting user tasks in the user-task inspector for possible regrouping (i.e., not an actual interface-building operation) and having the palette change with each selection (a true interface-building operation).

RELATED WORK

There are three areas directly related to our work: interface builders, user-centered design, and model-based interface development. There are excellent comprehensive surveys of existing interface builders and other software tools [3]. We will refer to those surveys but will remark again that we are not aware of any interface builder that exploits interface models to support its operations. Similarly, much has been written about user-centered design [4]. However, no specific implementations have arisen from this field to address the shortcomings of interface builders.

The work closest related to ours is that of other model-based interface development systems. UIDE [1] was one of the first systems to introduce the notion of using interface models to drive interface development. ADEPT [2] used effectively for the first time user-task models in their approach to generate user interfaces. UIDE [1] and Mecano [6], among others, exploited the idea of being able to generate automatically the elements of an interface layout from the attributes of the data/domain objects to be displayed on the interface. A number of other systems have also improved or modified to a certain extent the techniques of user-task modeling and interface generation.

The key difference between earlier systems and MOBI-D is that the former placed an emphasis on the automated generation of an interface given a partial interface model. For example, generating a concrete interface in ADEPT from a user-task model. Because of the automated approach, these systems did not attempt to incorporate interactive tools, such as an interface builder, directly into their interface modeling approaches. Therefore, efforts such as MOBILE have not been attempted in the past by those systems.

CONCLUSIONS

We have presented a software tool, called MOBILE, which enables user-centered design approaches for interface builders. The tool combines the recognized benefits of user-centered design with the efficient functionality of interface builders. We have, in addition, created knowledge-based techniques for decision support that further augment the capabilities of the tool over conventional interface builders. We have evaluated the tool

and made extensive changes to its design based on user recommendations.

MOBILE can serve as an initial step also in demonstrating the value of model-based interface development technologies. We expect to further enhance MOBILE by providing additional decision-support functions, such as layout critics. Our current experience with the tool is of course limited. We do not know yet how it will respond in designs that include large and complex user-task models. Nor do we have an extensive knowledge base of interface design guidelines that will cover a majority of widget assignment situations. However, we feel the MOBILE approach significantly helps in advancing user-centered design principles in practical user interface building, a definite worthwhile goal.

ACKNOWLEDGEMENTS

The work on MOBI-D is supported by DARPA under contract N66001-96-C-8525. We thank Hung-Yut Chen, James J. Kim, Kjetil Larsen, David Maulsby, Dat Nguyen, David Selinger, and Chung-Man Tam for their work on the implementation and use of MOBI-D.

REFERENCES

1. Foley, J., *et al.*, *UIDE-An Intelligent User Interface Design Environment*, in *Intelligent User Interfaces*, J. Sullivan and S. Tyler, Editors. 1991, Addison-Wesley. p. 339-384.
2. Johnson, P., Wilson, S., and Johnson, H., *Scenarios, Task Analysis, and the ADEPT Design Environment*, in *Scenario Based Design*, J. Carrol, Editor. 1994, Addison-Wesley.
3. Myers, B., *User Interface Software Tools.* ACM Transactions on COmputer-Human Interaction, 1995. 2(1): p. 65-103.
4. Norman, D. and Draper, S., eds. *User Centered System Design*. . 1986, LEA.
5. Puerta, A. and Eisenstein, J. *Interactively Mapping Task Models to Interfaces in MOBI-D*, in Proc. of *DSV-IS98: Eurographics Workshop*. 1998. Abingdon, England.
6. Puerta, A. and Eriksson, H. *Model-Based Automated Generation of User Interfaces*, in Proc. of *AAAI'94*. 1994: AAAI Press.
7. Puerta, A. and Maulsby, D. *Management of Interface Design Knowledge with MOBI-D*, in Proc. of *IUI97: 1997 International Conference on Intelligent User Interfaces*. 1997.
8. Puerta, A. R. A Model-Based Interface Development Environment. *IEEE Software*, (14) 4, July/August 1997, pp. 40-47.
9. Tam, R. C.-M., Maulsby, D., and Puerta, A. *U-TEL: A Tool for Eliciting User Task Models from Domain Experts*, in Proc. of *IUI98: 1998 International Conference on Intelligent User Interfaces*. 1998. San Francisco, CA: ACM Press.
10. Valaer, L. and Babb, R. Choosing a User Interface Development Tool. *IEEE Software*, (14) 4, July/August 1997, pp. 29-39.