

MANAGEMENT OF INTERFACE DESIGN IN HUMANOID

Ping Luo, Pedro Szekely and Robert Neches

USC/Information Sciences Institute and Department of Computer Science
4676 Admiralty Way, Marina del Rey, CA 90292
E-mail: {ping, szekely, neches}@isi.edu

ABSTRACT

Today's interface design tools either force designers to handle a tremendous number of design details, or limit their control over design decisions. Neither of these approaches taps the true strengths of either human designers or computers in the design process. This paper presents a human-computer collaborative system that uses a model-based approach for interface design to help designers search the design space effectively and construct executable specifications of application user interfaces. This human-in-the-loop environment focuses human designers on decision making, and utilizes the bookkeeping capabilities of computers for regular and tedious tasks. We describe (a) the underlying modeling technique and an execution environment that allows even incompletely-specified designs to be executed for evaluation and testing purposes, and (b) a tool that decomposes high-level design goals into the necessary implementation steps, and helps designers manage the myriad of details that arise during design.

KEYWORDS: Interface-Building Tools and Techniques, Design Processes, Development Tools and Methods, Rapid Prototyping, Interface Design Representation.

INTRODUCTION

Today's interface design tools do not strike an adequate balance between giving designers control over an interface design, and providing a high level of design automation. Tools that give designers extensive control over details of a design, like interface builders [14], typically force designers to control *all* details of the design. Tools that automate significant portions of interface design, like MIKE [15] and UofA* [16], typically let designers control *few* of the details. This paper describes our efforts to combine the benefits of these two classes of tools to provide extensive human control without drowning designers in details. Our thesis is that model-driven programming plus decomposition of design goals lets us provide an environment that allows humans and computers work from their strengths in the design process. In the system described below, humans focus on setting high-level policies, while the computer focuses on bookkeeping and details that humans do not want to address.

Our approach is motivated by a view of interface design as a

search in a space of design alternatives. This search space contains an unmanageable number of design alternatives. It is defined by design decisions such as determination of the operations to be supported and parameters they require, presentation of these operations and their parameters, presentation of application objects of interest, choice of input gestures, and control of sequencing among the gestures and operations. Given that there are many choices for each of these considerations, the design space explodes combinatorially. Thus, the key problem is to help designers search the design space quickly and effectively.

It is interesting to consider how interface builders and automated generation systems, the two main current approaches, fare when judged in terms of support for exploration of design alternatives.

Interface builders [13, 14] allow designers to draw the screens of an application. They are most effective for concrete tasks such as constructing the static portion of interfaces and editing screen layouts. But, working at this level, they force designers to handle far too much detail. To use the tools, designers are forced to make design commitments down to the level of individual widgets. This distracts them from design decision making at a conceptual level (e.g., committing to support selected objects in the interface without committing to how they will be presented). Furthermore, the number of steps entailed to change a high-level commitment discourages exploring many alternatives. Missing from interface builders are support for design abstraction and the ability to defer design commitments.

Automatic interface generation systems [2, 5, 7, 15, 16] generate user interfaces based on a description of application functionality. The main purpose of these systems is to hide interface design complexity by automating all design decisions made by human designers. Because of this, they provide human designers with very little control over those decisions. Unfortunately, generating good interface designs is intrinsically difficult. Thus, the resulting automatically-generated interfaces either are not good enough for real use [10], or have rather limited styles. (The best is the menu-driven style, which is limited to generating the dialogue portions of the display because the application display area is too hard to generate automatically.) The technical barriers to effective automation are hard to overcome. The design space is large and contains many bad designs. Principles of good user-interface design are not yet well specified [4], and cannot be used to automatically prune the search space. It is hard to

tell an automated interface design system about application-specific considerations that affect weighting of alternatives.

Our approach strives to retain the best aspects of both approaches. We allow designers to control design decisions like interface builders do, but with the additional ability to control decisions with respect to a broader set of concerns (e.g. presentation and sequencing, not just layout). We preserve the ability of automation tools to let designers insulate themselves from some design details, but provide hooks to control those details if desired. We facilitate design exploration by supporting different levels of abstraction to maximize modularity of the design space and control of its search. We also provide aids for managing the details of implementing conceptual designs. In this environment, human designers focus on design decision making and evaluation (which is very hard to automate but can be performed quite effectively by humans). The system helps designers reduce their memory burden and mental work load by managing regular and tedious tasks, such as decomposing designs into easily achievable sub-designs.

Our approach consists of two parts: a model-based design and execution environment [18, 19], and a human-computer collaborative design exploration environment. The model-based environment provides a modeling language for many aspects of interface design, ranging from the abstract considerations that arise early in the design, to the concrete ones that define the details of the interface [12]. The model-based environment also provides the run-time system to execute the modeled interfaces, even before they are fully concretized. The collaborative design environment helps designers effectively search the design space by helping them construct the models and by helping them keep track of: (a) alternative design alternatives about which they must make decisions and (b) consequences of those decisions which require handling.

The rest of the paper is organized as follows. We first briefly describe the enabling technology, a model-based system that provides capabilities for modeling and for design model execution. Next, we present the collaborative design environment in action using a design example to show: how we decompose design goals, how we divide labor between the designer and the computer, and how the human designer interacts with computer. We then close with our current status and plans for future work.

MODEL-BASED DESIGN & EXECUTION ENVIRONMENT

HUMANOID [19], the base layer of our environment, is a model-based system. Interfaces are specified by constructing a declarative model of how the interface should look and behave. The HUMANOID model provides the enabling technology for supporting different levels of abstraction to maximize the modularity of design.

The Interface Model

HUMANOID's design model captures information about an application's functionality (objects and operations), and information about all features of the interface. HUMANOID factors the model of an application and its interface into five semi-independent dimensions: *Application semantics design*

represents the operations and objects that an application program provides; *Presentation* defines the visual appearance of the interface; *Manipulation* defines the gestures that can be applied to the objects presented, or equivalently, the set of manipulations enabled at any given moment; *Sequencing* defines the order in which manipulations are enabled; and *Action side effects* declare the actions that an interface performs automatically as side effects of the action of a manipulation. (see [19] for details).

The Run-Time System

HUMANOID provides a run-time support module, which is included in every application. The module executes the design model; that is, it constructs application displays and interprets input according to the specifications in the model.

To produce or update the display of an application data structure, the run-time system searches the presentation model hierarchy for a presentation component capable of displaying the data structure. The model returns the most specific presentation component suitable for displaying the data in the given context (e.g. taking into account data type congruence and spacing restrictions) and the run-time system uses it to produce or update the display. Note that the presentation component obtained from the model might either be a default inherited from HUMANOID's generic knowledge base, or a more specific presentation component specified by an interface's designer. The defaults enable designers to execute incompletely specified designs for testing and evaluation purposes.

This provides HUMANOID the capability to support delay of design commitments which allows designers specify only those aspects that they want to address. Also, with the run-time system, developers do not need to write code to update the display. They specify declaratively the dependencies between presentation methods and application data structures. HUMANOID uses this information to dynamically reconstruct the displays when the application data structures change.

HUMANOID's Interactive Modeling Tools

The model-based approach in HUMANOID leads to an interface design environment with a number of benefits not found in current interface building tools [20]:

- *Supports design reuse.* The system provides an easy way to access all the design models by presenting components of the models as graphical objects which can be selected, dragged and dropped into a position for reference and reuse.
- *Helps to understand design models.* The environment presents the designer both the model and sample displays of the interface specified by the model. By using explicit models, HUMANOID can reduce designers' mental effort to understand the specification in two ways. First, it helps by showing designers connections between components of the model and components of the display. Second, it helps designers understand the impact of a potential design change by identifying where the altered portion of the model would be applied. For example, designers can point at an interface display object of interest and let

HUMANOID identify and fetch the part of the model that defines that object; designers can also ask HUMANOID to highlight all the interface objects on the display that are controlled by a particular part of the model (see [20] for more details).

- *Reduces the cost of modeling.* Although a model-based approach provides the benefits above, it does so at a price of additional specification effort. The design environment reduces the cost of obtaining these benefits by assisting with the burdens of building the model. The HUMANOID model are presented in a context-sensitive, graphical working environment which allows designers to build and modify the models interactively. Any modification of the model will force recomputation of the interface defined by that model, and thus provides immediate feedback. This eases the design of dynamic aspects of the interface, including conditional display, interactive behavior, and sequencing control, yet still retains the descriptive power of language-based systems.

COLLABORATIVE DESIGN ENVIRONMENT

This section uses an example to illustrate how the environment divides labor between human and computer to utilize what each does best in design. We first argue that design is complex in nature and we cannot hide complexity from designers without losing designers' control over design decisions. We then introduce an agenda system that we use for the bookkeeping needed to manage design tasks. Following that, we illustrate how we decompose design goals and how the designer interacts with the system to implement a design policy decision.

The Nature of Design: Where Support is Needed

To use any design tool that provides extensive control over the design, designers must know (a) how to break down high-level design goals into activities that are supported by the tool and (b) how to resolve interactions between these activities. Although the HUMANOID interactive design environment combines the strength of interface builders and model-driven programming, it is not immune to this problem. Decomposing an interface design is a difficult task (if it is doable at all by non-programmers).

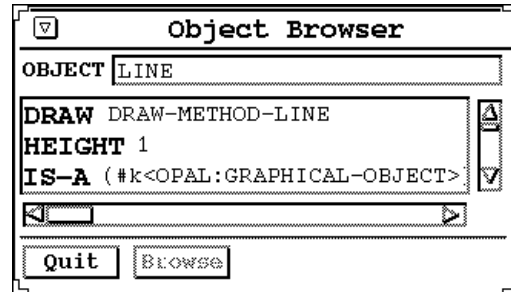
Since interface design is a rich domain, most high-level goals break down into multiple lower-level goals. There are often many ways in which to achieve the lower level goals, giving rise to complexity that distracts designers from current design issues by requiring effort to keep track of design steps and update agendas of activities. This is not an artifact of the model-based approach, but an intrinsic characteristic of design. One still must face decisions about exactly how to implement design policy decisions, regardless of whether one uses a model-based approach, interface builders or hand programming. All that differs between these paradigms is how explicitly those decisions are expressed, and therefore how much help the system can provide in effecting them.

The following example illustrates the complexities in what appears to be a simple interface design decision. Consider the interface for the object browser shown in Figure 1a. The

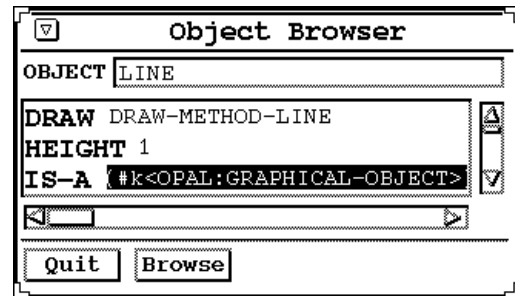
main body of the object browser window shows a list of slot-value pairs. The designer would like to make the values selectable so that commands can be applied to them. Figure 1b shows the object browser having the selection capability.

Figure 1. Object Browser Example.

(a) Browser with no notion of "current selection"



(b) Browser with support for selection



In general, to implement such a capability, one has to perform many tasks. For each task (which we label below for future reference) there can be many possible methods:

- *TASK-1.* Determine whether single or multiple objects can be selected.
- *TASK-2.* Specify or create a variable to record the current selected object (called an application input in the HUMANOID jargon). Once an object is selected, it must be stored in a variable that can be referenced by commands that use the selection.
- *TASK-3.* Define the manipulation that selects the object. This entails the following sub-tasks:
 - *TASK-3.1.* Choose the mouse button or key that invokes the selection behavior.
 - *TASK-3.2.* Determine the area of the presentation where the mouse can be clicked to select the object. It could be the complete presentation of the object, only a hot-spot within its presentation, or an area larger than the presentation (if the presentation is very small).
 - *TASK-3.3.* Specify the mapping from (a) the presentation area where the selection is triggered, to (b) the application object implied by the selection. In general, a presentation can display multiple components of an application data structure, and it is necessary to specify which one is the one to be selected.
- *TASK-4.* Design the feedback used to show the selected object. The feedback can be represented by an icon associated with the selected object (e.g. a check mark), or

by highlighting the display of the selected object using different shapes, colors, and filling styles.

There are also many potential interactions between the tasks just listed and decisions that may have been made elsewhere in the design. Some examples: (a) other behavior on the objects should not use the same mouse button as is used for selection; (b) the type of the variable where the selection is stored must be consistent with the type of the value to be stored in it; and (c) if highlighting feedback is used, it should not be confused with other possible highlightings used for other purposes in the same presentation.

Notice that these complexities involve fundamental design decisions. No matter what tools we use, we cannot hide the complexity without reducing the design space, which would unduly limit designers' control over design decisions (recall our argument on automatic interface generation systems in the introduction of this paper).

Since this complexity is inevitable, the question is one of how best to cope with it. Our approach is to ameliorate the problems by introducing a new division of labor between human designers and computers. To do so, we decompose design goals into system-supported collaborative tasks. We take advantage of HUMANOID's explicit models, and add another layer of modeling to describe interface design activities.

The main idea is to model the goals that an interface design must satisfy, and to model the methods for achieving the goals. Goals can be posted automatically by the system (e.g. "all application commands should be executable by end-users"), or they can be posted by the designer (e.g. "the objects displayed should be selectable"). Complex design goals, like the example above, are decomposed into simpler goals until they bottom out in goals that can be achieved by simple editing operations on the interface design (e.g. adding a menu-item to a display).

By modeling interface design in a goal-oriented way, it is possible to use a "goal management" system to help designers keep track of what goals still need to be achieved, to offer designers different methods for achieving goals, to keep track of which methods have been tried, and to warn designers about previously achieved goals that become violated by the solutions to other goals.

Similar to Framer [8], our collaborative environment seeks to avoid the problems of automating human design responsibilities and to overcome weaknesses of human cognitive limitations (e.g. short term memory). Unlike Framer, our system supports both system and designer initiated design goals. The system-initiated goals are implementation oriented, like those generated by the critics in Framer. Our designer initiated goals are more oriented toward design intentions, but designers have the choice to post implementation oriented goals as well. Our environment goes beyond Framer in the following additional respects:

- *It is an active collaborator.* The system can react promptly by deriving necessary tasks from the current design and updating its task agenda automatically in

response to interface model changes. As a side effect, this frees designers from potential limitations that might be imposed upon designers by our modeling tools. The designer can do modeling by any other means and still get a valid agenda from the system.

- *It helps in evaluating designs by providing prompt results.* With HUMANOID, changes to the model are immediately reflected in the sample display that is generated from the current design. This provides quick design-evaluation-redesign iteration and makes the relationship between models and the interfaces generated from the models more understandable.
- *It supports interleaving design tasks.* By making all current tasks available to designers, we let designers choose to concentrate on any issue they want, and allow them to carry out design tasks in parallel. Designers can shift focus among their design tasks freely, without losing track of their other design tasks and without losing the feedback provided by the derived interface.

The rest of this section is organized as follows. First we describe our goal management mechanism, called Scenarios/Agendas. Then, we briefly describe the goal decompositions relevant to our object browser design example. Following that, we present an example of how the system assists a designer in managing the details that arise in implementing design decisions in that example. We summarize the benefits of this approach at the end.

Scenarios/Agendas

Our goal management system performs design task book-keeping, i.e., it decomposes high-level design goals, and maintains and presents a valid task agenda. It is based on Scenarios/Agendas [11], a tool for building applications where users need to manage a set of activities over an extended period of time.

Scenarios/Agendas provides two constructs for modeling activities: goals and methods. A goal represents a desired state of design (e.g. the interface provides at least one way for the user to invoke any application commands). Goals can have one or more methods for achieving them (e.g. to make all commands invocable by the user, one method is to use a menu-bar with all the commands). Scenarios/Agendas supports three kinds of methods. *System methods* automatically run a procedure that achieves the goal. This provides a way to encode default behavior that does not require user involvement. *Interactive methods* prompt the user for some information and then invoke a procedure. *Decomposition methods* replace a goal with a set of simpler goals that the user or the system can pursue individually.

Scenarios/Agendas provides a default interface for managing a large collection of activities. The interface presents an agenda of the set of goals that are to be met, and allows the user to choose among all the applicable methods for accomplishing the goal. In addition, Scenarios/Agendas provides ways for filtering the agenda to focus on different types of goals (e.g. all "present-object" goals), goals that refer to particular types of objects (e.g. all goals that refer to a particular command), or goals created by a particular person or on particular dates.

Scenarios/Agendas monitors the relevant data objects in the domain. When the data structures change, the satisfying conditions of any affected goals are re-evaluated, and their status is updated appropriately on the agenda. Our collaborative environment makes use of this facility to monitor the interface design model, and keep the goal agenda up to date as the design evolves.

Decomposing Interface Design

The design goals and their decompositions, which model the interface design process, are used by our goal management system to derive the design task agenda.

We have identified a set of design goals and methods that can be used to decompose the design process for constructing a wide variety of graphical interfaces. (Appendix 1 contains a catalogue of the top-level interface design goals we have identified.) In our scheme, knowledge about these goals and their decomposition is built into the system, rather than being a concern of individual designers.

Goals can be defined in terms of sub-goals or primitive steps. All goals define queries to the design model to test the appropriate condition. Non-primitive goals are decomposable into other goals that are easier to achieve. Below, we show the decomposition of the “made an object selectable” goal that was discussed previously.

Goal *Made Object Selectable*

Subgoals:

Selection Manipulation Elaborated {for TASK-3}

Subgoals:

Single/Multiple Selection Specified {for TASK-1}

Application Variable Specified {for TASK-2}

Start Where Specified {for TASK-3.1}

Event Specified {for TASK-3.2}

Value To Set Specified {for TASK-3.3}

Selection Feedback Specified {for TASK-4}

The Design Environment In Action

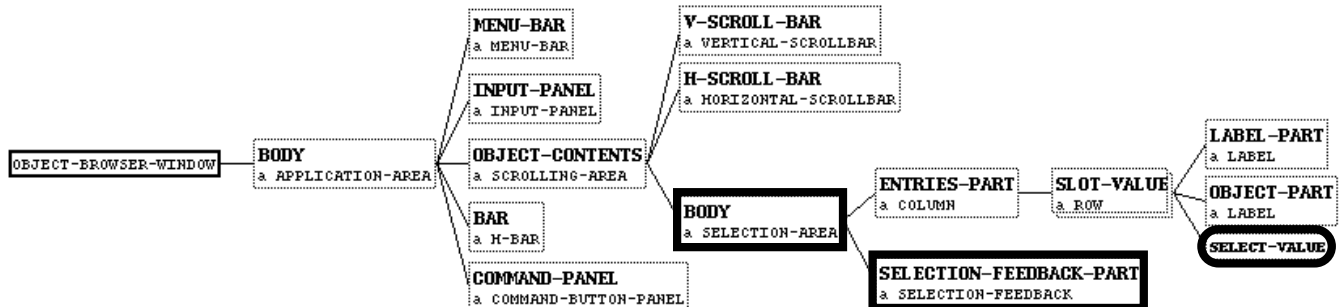
In this section, we show (a) how the system leads designers to map their conceptual designs into executable interface specifications; (b) how the system reduces burdens that designers otherwise have to bear; and (c) how the system

lets designers take control without forcing them to handle details that they do not care about. As an example, we show the system helping designers add selection capabilities to the object browser shown in Figure 1. The previous section showed the subgoals entailed by this goal. Figure 2 shows the enhancements to the design model required to specify this new interface feature. The elements that need to be added are shown with a thick border; they consist of new presentation model components labeled SELECTION-AREA, SELECTION-FEEDBACK and SELECT-VALUE. The figure does not show details of the parameters of the newly added design objects. The rest of this section describes the steps that the designer follows in modeling the desired new feature.

In the collaborative environment, designers express their intentions by selecting goals from the system’s design goal library and then posting instances of those goals to the agenda. The system also posts goals when it detects new design tasks. In our example of adding the select-object feature to the object browser, a designer selects the *Made Objects Selectable* goal from the goal library and, prompted by the system, specifies the display object to be made selectable. The latter is done by mousing on that object in the display of the sample implementation; the goal is then posted on the agenda. This collaboration lets designers keep control, because they can post goals at whatever level is appropriate; simultaneously, it lets the system help by taking responsibility for determining consequences of goals.

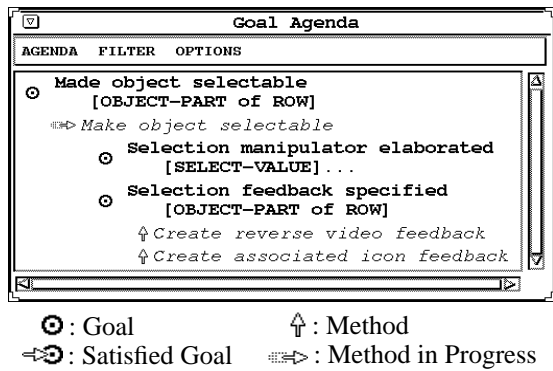
When the goal *Made Objects Selectable* is instantiated, the agenda system computes the goal decomposition and the necessary tasks from the design model in that context, and presents the agenda as shown in Figure 3. (The goals are shown in bold face with their parameters in square brackets.) Associated with each goal are all the known methods for achieving that goal. The methods will be displayed in the agenda when designers click on the icon to the left of a goal. Designers can select any method of any goal in any order, or they can edit the design directly using the HUMANOID direct-manipulation interface for editing the model. This approach lets designers choose among design alternatives without saddling them with the full burden of generating those options.

Figure 2. The presentation and behavior model of an object browser application. The browser shows object slots and values.



The display is defined by the OBJECT-BROWSER-WINDOW presentation method. Its body, APPLICATION-AREA, has five parts. Four of the them (MENU-BAR, INPUT-PANEL, COMMAND-PANEL, and HORIZONTAL-BAR) are generic to a wide class of application windows, and are inherited from the HUMANOID generic presentation model. The part SELECTION-AREA of OBJECT-CONTENTS is specific to our application for displaying slot-value information of a user specified object and for presenting feedback of the selected object.

Figure 3. Agenda after *Made Object Selectable* is posted, and *Made Object Selectable* and *Selection Feedback Specified* are expanded by designers. Goals are shown in bold face with their parameters in squared brackets. Methods are in italic. The indentation presents the decomposition relation.



In this case, as shown in Figure 3, the system provides two methods for achieving the *Selection Feedback Specified* goal. The *Create Reverse Video Feedback* modifies the design to highlight the selected object using reverse-video. The *Create Associated Icon Feedback* modifies the design to indicate the selected object using an icon (e.g. a check-mark) displayed next to the selected object.

Suppose the designer chooses the *Create Reverse Video Feedback* method. That method posts two new goals on the agenda: *Selection Area Specified* and *Feedback Presentation Elaborated* (Figure 4). Since both subgoals have default system methods, the system executes them. Accordingly, the icons next to the posted subgoals are changed to show that they have been satisfied. Notice that the environment utilizes several means to inform designers about effects of its default methods: (a) the model in Figure 2 changed to present that the default methods added a SELECTION-AREA and a SELECTION-FEEDBACK to the design; (b) the sample interface generated from the model was updated to reflect the presentations and behaviors of the new model; (c) the agenda presented the new status of the goals and methods by using different icons. At this point, the designer can still choose other methods and change the way in which the goal was satisfied by the system. This mechanism of providing defaults allows designers to off-load onto the system portions of design they do not want to be concerned about, while retaining control when they so desire.

If the designer had chosen the *Create Associated Icon Feedback* method, the system would post the corresponding goals and methods and lead the designer to realize the implementation of the *Selection Feedback Specified* goal differently. Knowledge about design inconsistencies is defined in the goal decomposition. For example, a multi-valued input needs a list of values not a single value. If the designer had chosen the single selection to realize the *Single/Multiple Selection Specified* subgoal and a multi-valued input to satisfy the *Application Variable Specified* subgoal, the

system would handle the bookkeeping tasks of posting the corresponding goals and methods to resolve the conflict.

Figure 4. Agenda after *Create Reverse Video Feedback* method is selected.

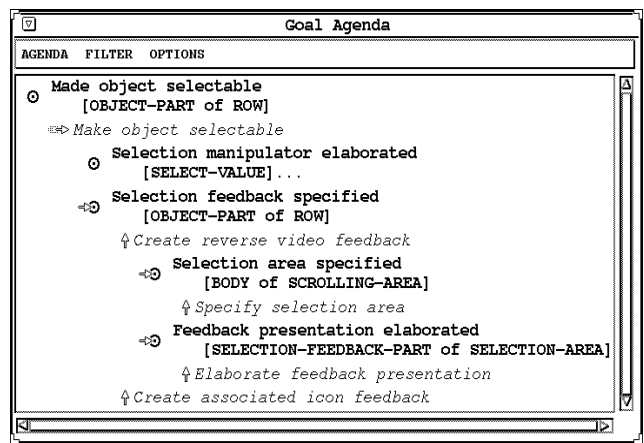
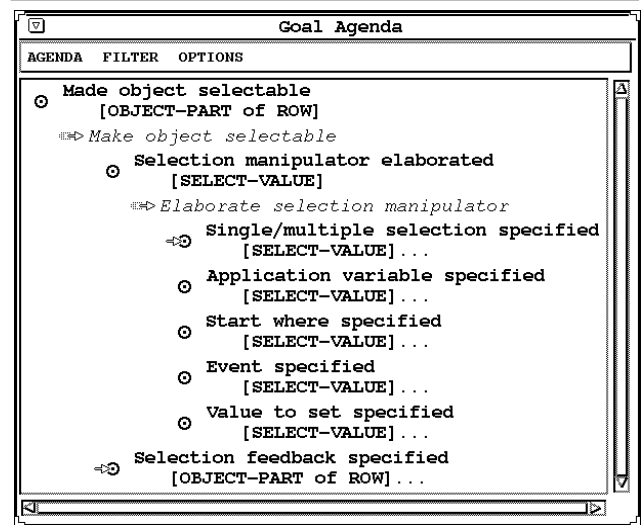


Figure 5 shows the subgoals of *Selection Manipulator Elaborated* that designers need to fill in the application-specific information. Satisfying the *Selection Manipulator Elaborated* goal adds the remaining elements to the design.

Figure 5. Agenda after *Selection Feedback Specified* is closed and *Selection Manipulator Elaborated* is expanded.



Benefits of the Collaborative Environment

In the collaborative environment, activity management is handled by the system. All the design options at each design step are presented to designers for consideration; there is no need for designers to worry about keeping track of all the steps needed to implement a top level goal. Equally important, the system does not impose any design commitments during the design process; designers have full control on design decisions. When tasks have been decomposed to the lowest level, the designer's activities are then supported by the HUMANOID modeling environment,

an effective system for detailed modeling.

The collaborative environment utilizes the strength of human designers for design decision making, and the strength of the computer for bookkeeping and automating regular and tedious tasks. More specifically, it has the following benefits and novel features:

- *Designers can express their design intentions explicitly.* Designers can express high-level design goals (e.g. making an object selectable or draggable). Our collaborative environment maps them into refinements to the design model (as shown in Figure 3-5), thereby overcoming cognitive difficulties in directly expressing desired design features in the modeling language.
- *Designers have extensive control over design decisions but are freed from cumbersome design details.* In the collaborative system, designers always have the freedom to pursue design issues in any direction they choose. They keep control of the aspects they care about, while off-loading onto the system things they do not want to be concerned about (by accepting default methods, e.g. achieving the Selection Feedback Specified subgoal in the design example). The system manages the multitude of details that arise during interface design (e.g. the task of specifying selection feedback in the design example) by providing agendas with management and filtering mechanisms, along with built-in capabilities for keep agendas up-to-date.
- *Designers can work outside the collaborative environment.* Designers can directly edit designs using the Humanoid interactive tools, or by editing a textual representation in a text editor. The goals affected by these changes will be automatically checked. If the modifications change the state of the design so that goals become satisfied, they will be marked as such. If the modifications introduce inconsistencies, the appropriate inconsistency resolution goals will be posted. However, the system does not do plan recognition to try to infer new goals to capture the high-level intentions behind the editing operations performed outside the environment. So, the environment can provide assistance when requested, and does not hinder designers when they do not require assistance.
- *The system smooths the transition from conceptual design to implementation.* Dichotomizing design activities as conceptual design and implementation has led to disjointed interface design tools addressing different aspects of the design problem (e.g. [3, 21] and [13, 14]). This disjointedness imposed a gap between conceptual design and implementation where no tool in the middle ground supports the mapping from conceptual design into functioning interfaces. The system described in this paper bridges that gap by mapping designers' intentions into executable interface specification. Thus it supports an intertwined design and implementation working environment [17], a previously neglected need.

CURRENT STATUS AND FUTURE WORK

The HUMANOID model-based design and execution

environment is implemented in Garnet [9] and CommonLisp. We have used it within our group to implement the interfaces for two large applications, a logistics analysis system (DRAMA), and a knowledge base development environment (SHELTER). In addition, the HUMANOID interactive design environment is itself implemented with HUMANOID.

We have done a theoretical study on identifying and classifying design goals, and have built the initial system described in this paper by adapting the current Scenarios/Agenda mechanism to support the interface design environment in full scale. We plan to perform usability tests to see how well the goals are understood and used by a wider range of designers and how well they lead designers execute interface design into functioning interfaces.

We also plan to provide design critics by integrating the kinds of design critics found in UIDE [5] into our system. The critics will detect design inconsistencies and provide some design evaluation automation. In our environment, the design critic would post a goal to remove the inconsistency. For some inconsistencies, system methods would fix the problem automatically and for others sub-goaling would be used to guide the designer to a solution.

ACKNOWLEDGEMENTS

We want to thank Peter Aberg, David P. Benjamin and Brian Harp for their helpful comments throughout the preparation of this paper. The research reported in this paper was supported by DARPA through Contract Numbers NCC 2-719 and N00174-91-0015. Contents represent the opinions of the authors, and do not reflect official positions of DARPA or any other government agency.

APPENDIX

The following is a summary of the top-level interface design goals and subgoals that we have identified up to now. These goals were identified by analyzing many of the interfaces that our group has built [1, 6, 11, 22] and also by analyzing interfaces of commercial products (e.g. FrameMaker) to understand how they would be modelled in HUMANOID.

Made-Command-Executable

Made-Command-Invocable

Made-Command-Invocable-By-Pulldown-Menu

Made-Command-Invocable-By-Button

Made-Command-Invocable-By-Popup-Menu...

Made-Command-Invocable-By-Dragging/Dropping...

Made-Command-Invocable-By-Keybaord-Accelerator

Had-Set-Method-For-Inputs

Set-By-Default-Value

Set-By-Dragging/Dropping...

Set-By-Selection...

Set-By-Type-In...

Set-By-Chosen (In Case Of Alternative Values)

Set-By-Dialogue-Box

Set-By-Factoring

Sequencing-Control-Elaborated

Executed-When-Read-To-Run

Reset-Inputs-After-Execution

Confirmed-Before-Execution

Shown-Dialogue-Box
Show-Dialogue-Box-When-Needed (Execute Otherwise)
Made-Command-Current-Selected
Made-Command-Default-Command

Has-Set-Method-For-Application-Inputs

Set-By-Default-Value
Set-By-Dragging/Dropping...
Set-By-Selection...
Set-By-Typing-In
Set-By-Chosen (In Case Of Alternative Values)

Made-Notified-When-Wrong-Value-Specified-For-Input

Beep-When-Incorrect
Revert-When-Incorrect
Message-When-Incorrect
Reset-When-Incorrect
Reinput-When-Incorrect
Error-Recover-When-Incorrect

Made-Object-Draggable

Dragging-Interactor (Behavior)-Elaborated
Start-Where-Specified
Event-Specified
Value(s)-To-Set-Specified
Command-To-Invoke-Specified
Command-Input(s)-To-Set-Specified
Feedback-Template-Elaborated

REFERENCES

- 1 P. Aberg and R. Neches. Clarification Dialogs: More Than Just Appearances For User Interfaces. *ISI working paper*.
- 2 W. Bennett, S. Boies, J. Gould, S. Greene and C. Wiecha. Transformations on a Dialog Tree: Rule-Based Mapping of Content to Style. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, pp. 67-75, November 1989.
- 3 J. Conklin and M.L. Begeman. gIBIS: A Hypertext Tool for Exploratory Policy. In *Proceedings CSCW'88*. September 1988, pp. 140-152.
- 4 S. Draper and D.A. Norman. Software Engineering for User Interfaces. *IEEE Transactions on Software Engineering*. pages 252-258. March 1985.
- 5 J. D. Foley, W. C. Kim, S. Kovacevic and K. Murray. UIDE: An Intelligent User Interface Design Environment. In J. S. Sullivan and S. W. Tyler, editors, *Intelligent User Interfaces*. pp. 339-384. ACM Press, 1991.
- 6 B. Harp, P. Aberg, D. Benjamin, R. Neches, P. Szekely. DRAMA: An Application of a Logistic Shell. *ISI/RR-91-284*. March 1991
- 7 P. J. Hayes, P. Szekely and R. Lerner. Design Alternatives for User Interface Management Systems Based on Experience with COUSIN. In *Proceedings SIGCHI'85*. April 1989, pp. 169-175.
- 8 A. C. Lemke and G. Fischer. A Cooperative Problem Solving System for User Interface Design. *Proceedings of AAAI-90*, pp.479-484.
- 9 B. A. Myers, et. al. Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces. *IEEE Computer* 23(11), pp. 71-85, November, 1990.
- 10 B. A. Myers. State of the Art in User Interface Software Tools. In H. Rex Hartson and Deborah Hix. Ed., *Advances in Human-Computer Interaction, Volume 4*, Ablex Publishing, 1992.
- 11 R. Neches, D. Benjamin, J. Granacki, B. Harp, and P. Szekely. Scenarios/Agendas: A Reusable, Customizable Approach to User-System Collaboration in Complex Activities. *ISI working paper*.
- 12 R. Neches, J. Foley, P. Szekely, P. Sukaviriya, P. Luo, S. Kovacevic, and S. Hudson. Knowledgeable Development Environments Using Shared Design Models. *The 1993 International Workshop on Intelligent User Interfaces (IWIUI'93)*. January 4-7, 1993.
- 13 Neuron Data, Inc. 1991. *Open Interface Toolkit*. 156 University Ave. Palo Alto, CA 94301.
- 14 NeXT, Inc. 1990. *Interface Builder*, Palo Alto, CA.
- 15 D. Olsen. MIKE: The Menu Interaction Kontrol Environment. *ACM Transactions on Graphics*, vol 17, no 3, pp. 43-50, 1986.
- 16 G. Singh and M. Green. A High-level User Interface Management System. In *Proceedings SIGCHI'89*. April 1989, pp. 133-138.
- 17 W. Swartout and R. Balzer. On the Inevitable Intertwining of Specification and Implementation. *CACM* 25, 7 (July 1982), pp. 438-440.
- 18 P. Szekely. Template-based mapping of application data to interactive displays. In *Proceedings UIST'90*. October 1990, pp. 1-9.
- 19 P. Szekely, P. Luo, and R. Neches. Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design. In *Proceedings SIGCHI'92*. May 1992, pp. 507-515
- 20 P. Szekely, P. Luo, and R. Neches. Beyond Interface Builders: Model-Based Interface Tools. In *Proceedings INTERCHI'93*. April 1993.
- 21 Y.Y. Wong. Rough and Ready Prototypes: Lessons from Graphic Design. In *Poster and Short Talks of CHI'92*. May 1992, pp.83-84.
- 22 J. Yen, R. Neches, M. Debellis, P. Szekely, and P. Aberg. BACKBORD: An Implementation of Specification by Reformulation. In J. S. Sullivan and S. W. Tyler, editors, *Intelligent User Interfaces*. pp. 421-444. ACM Press, 1991.