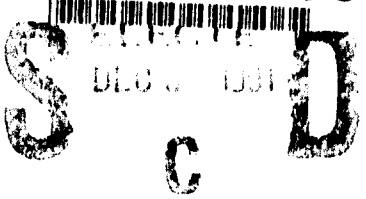
TRW Systems Engineering &
Development Division



TRW ①

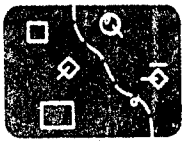
TRW-TS-91-02

A System for Specifying and Rapidly Prototyping User Interfaces

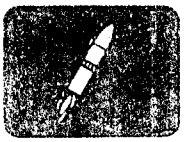
Christopher Rouff

June 1991

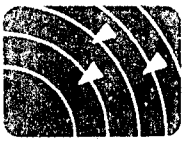
TRW Technology Series



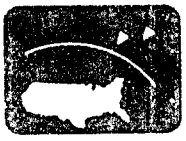
TRW Technology Series



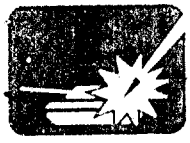
TRW Technology Series



TRW Technology Series

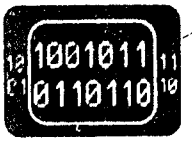


TRW Technology Series

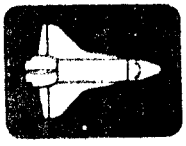


TRW Technology Series

TRW Technology Series



TRW Technology Series



chnology Series

A System for Specifying and Rapidly Prototyping User Interfaces

Christopher Rouff
TRW Inc.
Systems Engineering & Development Division
Carson, California

Ellis Horowitz
Computer Science Department
University of Southern California
Los Angeles, California

ABSTRACT

This paper presents a system for rapidly prototyping user inerfaces and a model for specifying them. The model represents the components, flow of control, constraints, and semantic feedback of an interface while the system allows the interface layout, dialogue, and application interface to be interactively defined, resulting in little or no programming to produce a prototype. As a user interface is prototyped, a model of the interface is maintained that represents its structure and functionality. This model can then be compiled into source code with calls to the X Windows and MOTIF libraries. The generated program can then be compiled and linked to application functions to produce an executable program.

Other user interface builders only have interactive definitions of the interface layout and specify the dialogue in a programming language, thus requiring a programmer to learn yet another language. The system described here is based on a graph model instead of a programming language and allows complex sequencing, constraints, and the application interface to be defined interactively. This graph model representation also allows analysis of the interface to be made in terms of consistency and completeness of the specification.

I. INTRODUCTION

Rapid prototyping of user interfaces allows a designer to produce a proposed interface in a short time, to easily experiment with different approaches to the interface, and to allow end users to try it early in design, when it is most cost effective to make changes [2]. The primary hindrance of producing a prototype is the amount of code needing to be handwritten. Reducing the programming reduces the designer's reliance on a programmer, decreases development time, and leaves the interface more modifiable (which supports iterative design).

We are currently developing a methodology and User Interface Management System (UIMS) [6] for prototyping user interfaces called Reduced Programming Prototyper (RPP). This system and methodology allow a designer to:

- interactively lay out the graphical components of the interface,
- combine components into hierarchical groups,
- specify the control flow between interface components and groups,
- define constraints between interface components, and
- define semantic feedback between the interface and the underlying application,

without writing any code. As the interface is being constructed, its components, flow of control, constraints, and semantic feedback are represented by a formal model, which can be used as the interface specification. This specification can be compiled into C code with calls to the OSF/MOTIF X Window toolkit [11].

Figure 1 shows the structure and components of RPP. An interface designer interacts with RPP through an interface editor. As the designer draws the components and defines relationships between them, the definition of the interface is saved onto a formal structure, called an interface representation graph (IRG), which can be checked for inconsistencies and incompleteness. The designer produces an executable version of the interface code by issuing the generate command, which causes the interface compiler to read in the interface representation and compile it into C code with calls to the MOTIF toolkit and X library. At this point the source code is compiled and linked with any user-written functions, which then can be executed by the designer.

In the remainder of this paper we discuss related work, discuss the components of an interface, describe the methodology a designer follows to specify the interface, and define the formal representation of the interface components and their interaction.

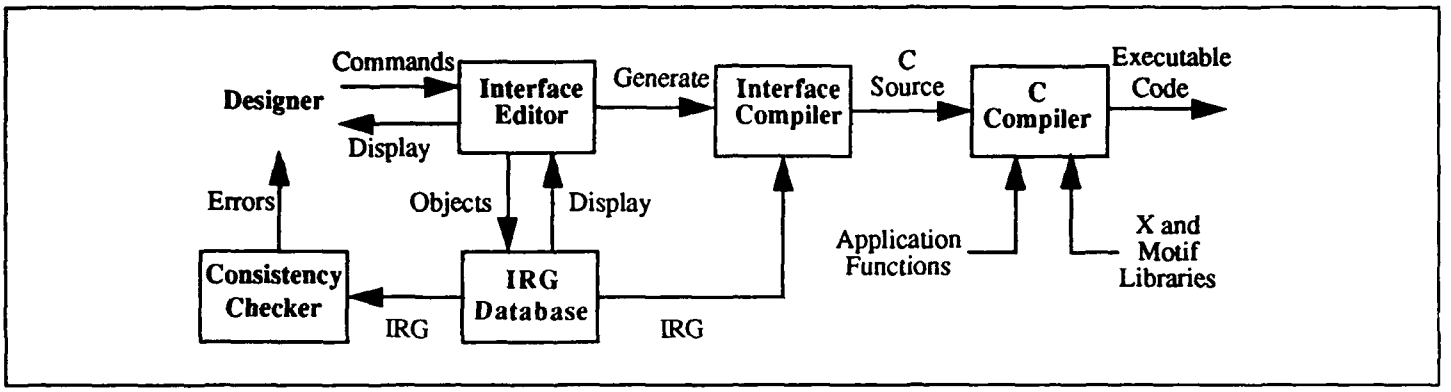


Figure 1. Components of RPP.

II. RELATED WORK IN PROTOTYPING TOOLS

There are three basic types of prototyping [18]: storyboards or slide shows, Wizard of Oz simulations, and testable simulations. The storyboard or slide show is a predefined set of screens displayed in a predetermined order, the Wizard of Oz prototype uses a person behind the scenes to drive the interface, and the testable simulation is a fully functional user interface that can be tested by the end user. RPP addresses the testable simulation type of prototyping. Production of this kind of prototype requires the largest investment in time and money so it is often integrated into the final product [3]. Current tools for producing testable simulation prototypes can be divided into three groups: User Interface Management Systems (UIMSs), Interactive Design Tools (IDTs), and high-level language-based systems.

A UIMS's primary goal is to separate the user interface from the application. It can be defined as a system that provides a component for the presentation of the visual part of the interface, a component for the definition of the dialogue between the user and application, and a component for defining the interface between the UIMS and the application program [6]. The separation of the interface from the application allows different interfaces to be tried without modifying the application. Two current UIMSs are Serpent [1] and TeleUse by TeleSoft. Serpent and TeleUse both provide an interactive design tool for defining the layout of an interface, a dialogue language (Slang in Serpent and D in TeleUse) for describing the flow of control of the interface, and a specification language for defining the interface between the UIMS and application. Both systems use X Windows [16] for presenting the visual component.

IDTs are similar to interactive graphics editors but allow users to place interface objects such as buttons, lists, menus, and icons on the screen, instead of graphics objects such as circles and lines. Many IDTs also allow the designer to specify application functions to be executed when a particular input occurs (e.g., a button press). Most systems produce source code definitions in a programming language or layout description language that represents the screen layout. The dialogue portion of the interface is then written in a programming language, or if part of a UIMS, in a dialogue language that uses the earlier generated definitions. A large number of IDTs have been written, often as part of UIMSs or other development systems. Some of these systems are Dialog Editor by Cardelli [4], Interface Builder by NeXT [8], Prototyper by SmethersBarnes [15], XBuild by Nixdorf Computers [19], and Dev Guide by Sun Microsystems.

HyperCard [5] and OSU [9] also allow interface objects to be interactively laid out on the screen but differ from the other systems in that a limited amount of sequencing can be specified. In HyperCard the designer defines a series of cards with common backgrounds and differing foregrounds and links them together through button presses and other interactions. In this way, simple flow of control between cards can be defined without the need for any programming, though the Hypertalk scripting language can be used for more advanced schemes. OSU allows sequences of Macintosh interface objects to be specified based on user inputs. Peridot [13] is another system that lets a designer interactively lay out and construct interface objects by demonstration.

The third type of prototyping systems is based on high-level programming languages that have specific constructs for displaying and manipulating user interface objects and usually have capabilities to call functions written in a conventional programming language. Examples of these types of systems are Winterp [10], Garnet [14], and Tooltool [12]. Winterp is based on XLISP and the MOTIF widget set. It allows easy definition of widgets in X Windows and has the power of programming in LISP. Like Winterp, Garnet is LISP-based and provides a high-level interface to X Windows. Garnet also provides mechanisms for defining constraints between objects as well as object manipulation. Tooltool is based on SunView and acts as a mediator between the windowing system and an application. The designer defines what interface objects will be used and maps inputs to these objects into inputs for the application and maps outputs from the application onto the user interface.

RPP differs from these systems in that it allows the definition of complex sequencing of the dialogue, constraints, and semantic

feedback interactively without using a dialogue programming language. The application still needs to be programmed, but for prototyping purposes a mock-up facility is supported that simulates the call and return of application functions, allowing the application routines to be simulated. Previous prototyping systems have relied heavily on programmers to code any complex sequencing of the interface and in many cases to also define the layout of objects. With the exception of Peridot and Garnet, none of the systems allows the designer to define constraints between objects of the interface and none of the systems allow hierarchies of interface objects to be grouped together to reflect visual and flow-of-control similarities.

III. COMPONENTS AND INTERACTION OF AN INTERFACE

To produce a prototype with a minimal amount of programming, an underlying structure that describes the components and their interrelations needs to be defined. The building of this structure is then reflected in the methodology that a designer follows to produce an interface. We define a user interface to consist of:

- graphical objects,
- flow of control,
- semantic actions, and
- constraints.

The graphical objects are made up of the graphical images that are visible to the user, the control portion defines an ordering on the appearance of the objects, the semantic actions define how the interface interacts with the application, and the constraints define dependencies between the objects. The following sections discuss each of these parts.

A. Graphical Objects

The graphical objects consist of the buttons, menus, dialog boxes, text, and other items that are directly visible to the user. These objects can be divided into two groups: those that contain other objects and those that are atomic. The container objects are called windows and the others, graphics. Graphics that appear at the same time inside a window are grouped together into frames. When control is transferred from one frame to another, the content of the window is cleared and a completely new set of graphics is drawn. An example of a series of frames is a transaction-based system, such as a data base browser, in which the user fills in query fields, and the results are displayed by erasing the query and displaying the results in its place. Since modifications to a frame sometimes need to be made without erasing the entire contents of a window, a second type of frame is defined, called a subframe. Modifications to windows can consist of (but are not limited to) display of dialog boxes, addition of menus, lists, error messages, or help.

Some user interfaces are a series of frames, others are a single frame with subframes that continually modify the original frame, and others are a mix of these. An example of a frame-oriented user interface is a transaction-based system where data in fields of a form are filled, with new frames and fields appearing depending on the items selected or the values entered. An example of a single frame-oriented interface is a graphics editor, in which a single frame is displayed and changes are continually made to it. Most systems contain both types of frames. There may be several frames that make up the interface but are modified with dialog boxes, errors, and help messages that continually update or modify the original frame.

B. Flow of Control

Flow of control defines how the interface will change on inputs from the user or values returned by the application and represents an ordering on the interface objects over time. There is also an implicit flow of control defined by the windowing system (in this case X) that allows the input focus to be changed from window to window and graphic to graphic. This usually takes the form of moving the cursor from one window or graphic to another with the window or graphic that the cursor is over being the object that receives input. Control among frames and subframes is explicitly defined since it requires the interface to be modified.

C. Semantic Actions

Semantic actions are the operations that drive the application. These actions are tied to the events that are associated with the flow of control. When an event occurs, a semantic action can be performed. These actions can take the form of calls to user written functions, programs, or file input/output. These actions can also reference or set variables that represent attributes of graphics being displayed. This allows the user to pass data to the application and allows the results of calculations in the application to be printed or change the display.

D. Constraints

Constraints occur when an attribute of an interface object becomes dependent on a value computed in the application or the attribute of another object. Constraints are one way that an application displays its output. Examples of attributes that an application might set are the label, position, size, or color. These attributes can reflect values computed by the application. The second type of constraint occurs when an attribute of one object is dependent on the attribute of another. Examples are an object's position depending on another (e.g., when one object moves, the other follows), and objects sharing a color.

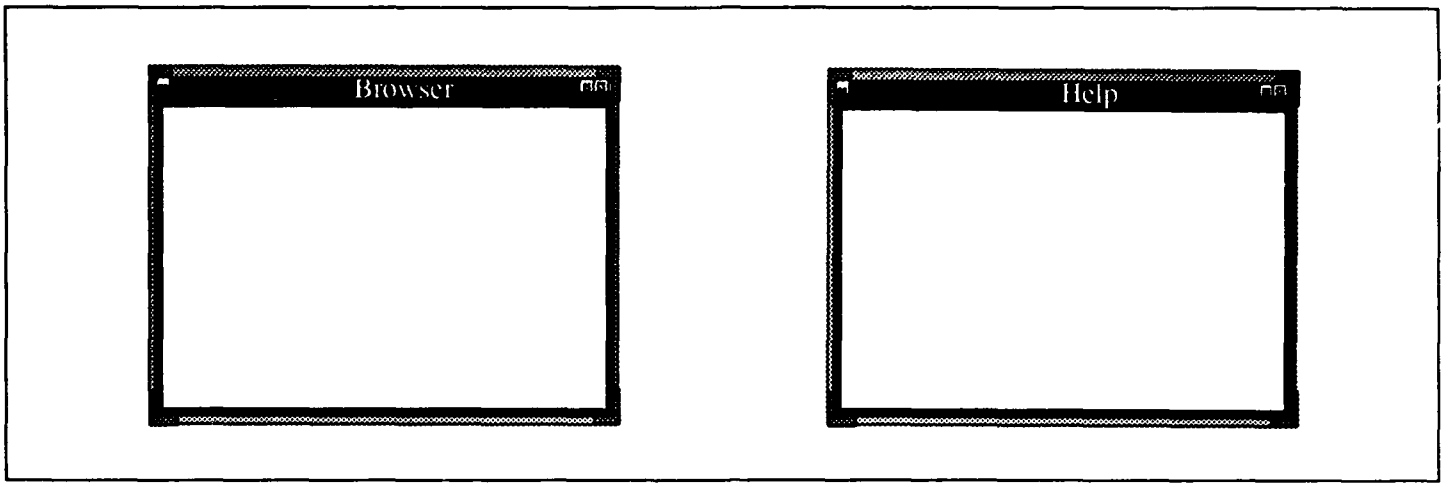


Figure 2. Two Top-Level Windows.

IV. PROTOTYPING AN INTERFACE

To prototype a user interface, a designer follows these steps:

1. The windows that are to appear in the interface are drawn.
2. Frames that are to appear in each of the windows are drawn.
3. The sequence in which the frames are to appear are defined.
4. Events are specified that determine when a sequence is to be taken.
5. Constraints between the components are defined.
6. Semantic feedback among the interface, the application, and the constraints are defined.
7. The specification is converted to C and X Windows code.

All interface objects are drawn in an interactive editor that allows the designer to place objects on the screen exactly how they will be seen at execution time. At any point in these steps the designer can repeat or reduce previous steps. This allows mistakes to be corrected, enhancements to be made, or different techniques to be tried, thus supporting iterative design. The following sections describe each of these steps in more detail.

A. Defining Windows

A designer begins a prototype by defining the windows in which the interface is to appear. A window can be defined as either a top-level window or a subwindow. Multiple top-level windows can exist and each window can have multiple subwindows. Appearance times for windows can also be specified, which define when the window will be displayed. The appearance time can be the same as the parent's or when control is transferred to the object. If it is the same as the parent's, then whenever the parent appears, the object also appears. If the window is a top-level window and does not have a parent, then it appears at the start of the application. Appearance when control is transferred to an object is how pop-up windows, dialogs, and other transitory windows are defined.

When specifying a window, its initial position and size are given by the designer. If it is a subwindow, its parent window is selected from the already defined windows. Next defined are the attributes of the window, such as the width of its border, its dimensions, and its appearance time. Figure 2 shows two windows after specification. They are both top-level windows, the left window (Browser) being the main window and the right window displaying help information.

B. Defining Frames and Subframes

After a window is defined, a series of frames is drawn to represent the appearance of the interface at different points in time. Frames can be defined in one of four ways: by the designer explicitly defining the graphics that make up the frame, by the designer defining a frame to be a modification of a previous defined frame (subframe), through semantic feedback from the application, or by a dependency on another graphic.

Hierarchies of windows, frames, or graphical objects can be defined by grouping together those sharing common attributes such as graphical objects, flow of control, or attributes (e.g., color). Groupings allow common attributes to be defined once for the entire group instead of individually for each item. A group of objects can be used like any other object; they can be listed as the source or destination of a sequence, graphics or attributes can be assigned to them (which are persistent across all frames and windows in that group), semantic feedback can be associated with them, and they can belong to other groups. Grouping similar objects together produces a consistent user interface by providing these benefits:

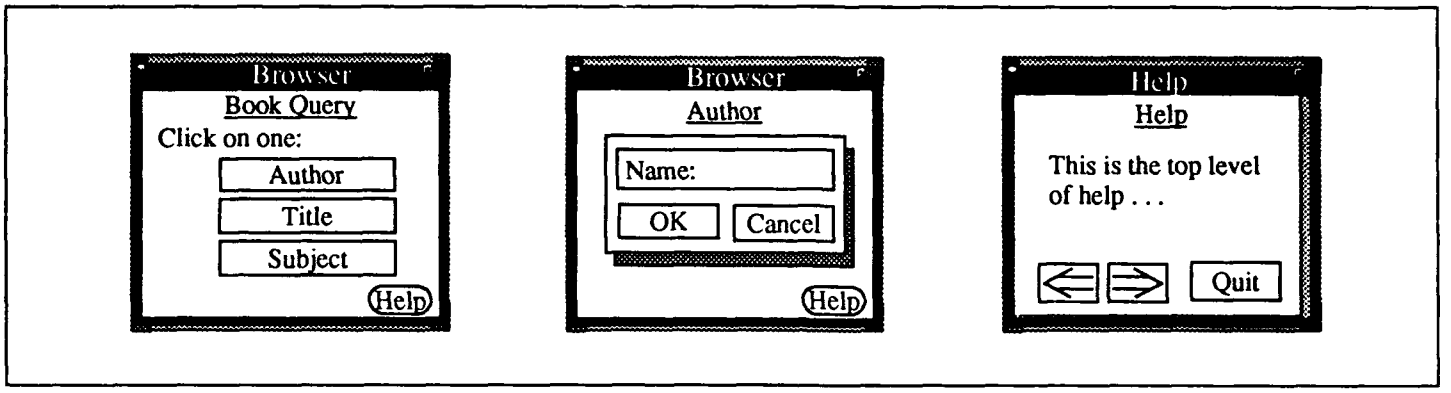


Figure 3. Three Frames of the Bookstore Query System.

- Persistent objects only need to be defined once.
- Persistent objects appear in the same place and look the same over the frames in which they appear.
- Identical flow of control properties need be defined only once.
- Objects with the same flow of control operate similarly.
- Modification of a shared object or sequence only has to be done once.

Figure 3 shows examples of frames that could be defined in the two windows shown in Figure 2. Two frames are shown in the Browser window and one frame in the Help window. The frame on the left in the Browser window is the initial frame listing a choice of queries that can be made. The center frame is the frame displayed if the first query is taken. The third frame (right) is the first frame of the Help window that is displayed whenever a help button is pressed. All the frames of the Browser window could be grouped together so the help button would only have to be defined once, causing it to appear in each frame of the Browser window in the same place and operate in the same manner, thus ensuring consistency in the look and operation of the button in all the frames of the browser.

C. Defining Flow of Control between Components

Once the windows, frames, and subframes are specified, the flow of control of the interface is defined by ordering the frames. The ordering consists of a set of sequences, each having a source and destination object. The source object can be a graphic, frame, subframe, window, or group. The destination of the sequence can be a frame, subframe, group, or window. Each sequence is assigned a value indicating what input initiates it. Input can be in the form of keystrokes, mouse button clicks, mouse movements, a value read from a file, or a value returned from a user-written function. Allowing inputs from the user or outputs from a program to initiate flow of control permits the interface to be either user-driven, application-driven, or a combination of both. Actions — graphical operations, user-written routines to be executed, values read or written to files — also can be assigned to sequences, which are executed when the sequence is taken.

D. Defining Constraints on Objects

Constraints can be defined on objects or flow of control of the interface. Constraints among objects (either windows or graphics) can be defined by declaring that a value of an attribute of one object is dependent on an attribute of another object. This is useful for defining such things as shadows, groups of objects whose positions are defined relative to one another, and objects with equivalent attributes such as the same foreground color. When defining an attribute of an object to be defined relative to an attribute of another object, a simple formula can be applied to the original attribute to transform it to a value for the new object. This transformation can be done by selecting the appropriate attributes of each object whose relationships are to be maintained or by explicitly naming the attributes involved and the transformation to be applied to those attributes.

Constraints on the flow of control can be defined by specifying events in multiple objects that need to occur before a sequence can be taken. An example is when a user has to select an item from multiple menus before the next frame is displayed or a value must be entered into a text field and a button pressed before execution can continue. Constrained and unconstrained flow of control can occur in the same frame, thus allowing the user to get help, quit, or skip the frame.

E. Defining Feedback to Objects

Input may be passed from the interface to the application, and the results returned to the interface for display or for modification of the interface appearance. Passing values between different parts of the interface allows a value input to one graphic to affect a second graphic without going through the application. A simple transformation formula can also be applied to these values to account for offsets. Values can also be read from files as well as passed from the application. This allows quick mock-ups of the

interface before the application is written and removes the need for a programmer from the early stages of the interface design. Values can also be saved to files to record data values entered by the user or other useful information. The values passed to the application are not restricted to input but can be attributes of a widget, such as position or dimension.

F. Generating the Interface

When the designer wants to try out the interface, the interface specification can be converted to User Interface Language (UIL) [11] and C code with calls to MOTIF and X Window routines, compiled, and then linked with the application functions. The UIL defines the initial appearance of the interface objects, the C code implements the flow of control of the interface, and modifications are performed through calls to MOTIF and X toolkit routines. Values defined as semantic feedback are stored in shared data structures, which can be read and written to by application functions. When the shared data values are changed, the corresponding attribute of an object is updated.

V. INTERFACE AND INTERACTION REPRESENTATION

As an interface is specified, it is mapped onto an IRG, which represents its structure, flow of control, interrelationships, and semantic feedback. This diagram models the structure and interaction between the graphics and application. The interface objects themselves are not addressed because it is assumed that the interface designer was given a predefined set of objects (such as the MOTIF widget set) and has no control over their functionality, only in how they are used. Such low-level actions as highlighting menu items are defined by the object and are not addressed in this model. The IRGs are not seen by the designer but are what drive the prototyping process. The designer only sees the actual objects in the editor. The following sections describe IRGs and how interface specifications are mapped onto them.

A. Representing Components of the Interface

IRGs, which are based on statecharts [6,16], can be defined as hierarchical transition diagrams in which nodes either stand for themselves or contain subdiagrams. It is this hierarchy of nodes that is used to represent the levels of windows, frames, and graphical objects of an interface. An interface is mapped onto the node structure by first representing the entire interface as a root node of an IRG, the top-level windows as children of the root node, subwindows as child nodes of the window's parent, frames as children of window nodes, and graphics as children of frame nodes. If a window contains subwindows, then the window has two types of children: frames and a subwindow. Grouping of nodes (such as frames or graphics) is represented by enclosing them in superstates, with the group possibly spanning across windows, frames or other groups. Groups are treated like any other node. They can have transitions, semantic feedback, constraints, or other attributes that apply to all the nodes inside the group.

There are other types of nodes. Default nodes are the nodes when execution starts in a window or group. A history node remembers the last subnode executed when the flow of control is temporarily interrupted. A diversion is a group of nodes that are executed as an aside to the current flow of execution, and a termination node stops execution of the application.

B. Representing Sequencing between Components

The sequences in an interface are represented by transitions that connect nodes of the IRG. Transitions are defined as tuples containing a source and a destination node, a value indicating when the transition can be taken, and actions to be executed. The circumstance for a transition to be taken, the semantics associated with a transfer of control, and the actions executed when the destination is reached, all depend on the types of nodes involved. If a window is the source of a transition, then whenever control is in the window and the value listed on its transition occurs, then the transition is taken and control is given to the destination node. If a frame is the source of a transition, the transition is taken whenever the listed input occurs among the frame's graphics. If a graphic is the source of a transition, then whenever the listed input occurs inside, the graphic control is transferred.

The type of a destination node determines the result of the transition. If the destination is a window node, then control is transferred to the window and all inputs are applied to the nodes in that window until the window loses control. The window must first transfer control to its start state: a group, frame, or graphic. If the distinction is a group, then, like a window, its start state is given control. If the destination is a new frame, then the graphics in that frame are displayed in the window; if it is a modified frame, the graphics are modified. Finally, if the destination is a graphic, such as a dialog box, then only that graphic can receive input. This forces the user to acknowledge a message or input a value before continuing with any other part of the interface.

Unlike transition diagrams and statecharts, IRGs do not require nodes to have transitions entering and leaving them. This is due to the fact that not all objects may change the state of the interface and not all receive input. An example of the former is radio buttons, which simply may set an application variable, and an example of the latter is a label.

An example of transitions between different interface components is shown in Figure 4. In this example, there are two windows. Window 1 has two frames and Window 2 has one frame. The default start node for Window 1 is Frame 1. This is where execution begins when control is given to Window 1. When Frame 1 receives control through the transition, it will display objects Obj 1 and Obj 2. Obj 1 does not have any transitions entering or leaving it. It could be a

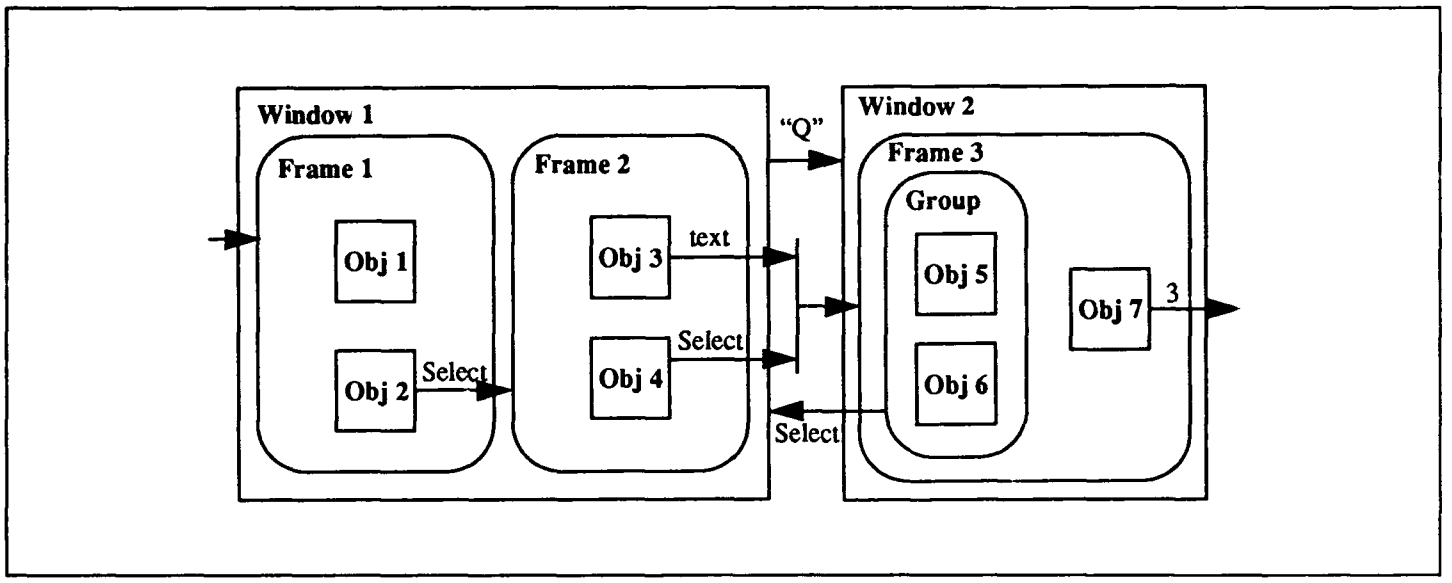


Figure 4. Transitions between Different Components of an Interface.

radio button label widget, or an object whose sensitivity has been turned off, like an invalid menu item. Obj 2 has a transition leaving it with a destination of Frame 2. This means that whenever the event "select" occurs in Obj 2, then Frame 2 is drawn. Frame 2 has an example of constrained flow of control. Control can only be transferred from Frame 2 to Window 2 when text has been entered into Obj 3 and Obj 4 has been selected. Window 2 has an example of a group of objects (Obj 5 and Obj 6) that might share some attribute or common flow of control.

C. Representing Semantic Feedback

Semantic feedback between the interface and application is represented through data feedback transitions (DFT) and application feedback transitions (AFT). These transitions show flow of data instead of flow of control. A DFT transfers data from a user interface object supplying a value to a parallelogram representing a variable containing data value shared between the interface and the application. An AFT has as its source a shared data value and as its destination a node representing the widget that is to be affected by the data. The values that flow between the interface and the application can be attached to any attribute of a widget, such as its position, size, label, or ability to accept user input (stippling).

D. Representing Constraints on Components

Constraints between widgets are represented similarly to the above semantic feedbacks. An interface constraint transition (ICT) has as a source the node representing the widget that is to be constrained, and as a destination the node representing the object to which the source is being constrained. An equation is associated with the transition that represents the transformation of the source widget's value to the destination widget's value. Constraints can be used on widgets to force them to maintain positions relative to each other or for one widget to display an attribute of another.

An example of semantic feedback and constraint between widgets, with the resulting IRGs, is shown in Figure 5. On the left side of Figure 5, a scrollbar and a text widget with a shadow are shown. The text widget displays a value relative to the location of the slider in the scrollbar. The right side of Figure 5 is the IRG representing the widgets. The square labeled "variable" represents a value shared between the interface and the application. The transitions from the scroll node and shadow node to the text node represent constraints. The transition between the text node and the variable node represents the data feedback, and the transition from the variable to the text node represents application feedback. Additional constraints between the scrollbar and the text widget could be defined to keep the widgets a fixed distance from each other. In such a situation, one widget's position would be constrained by the other via an offset added to the first's position.

E. Example of a Representation

Figure 6 is the complete IRG of the Browser interface described in the second section. The interface being specified is for a system that allows patrons of a bookstore to query the availability of books. At any time the user can move from the Browser window to the Help window, page through the Help window, and return to the Browser. Subsequent movement to the Help window resumes execution where it left off.

The outer node (Bookstore Query System) contains the entire interface description. The two states (Query and Help) in the root state represent the two windows of the interface. The subnodes in the Query node represent the frames that appear in that window and

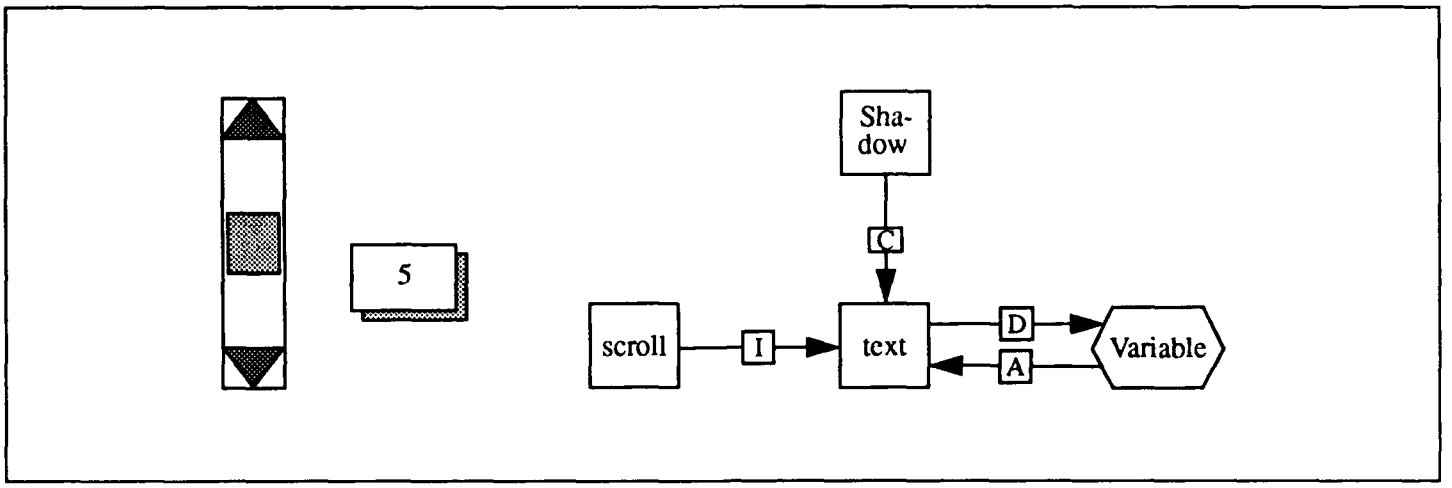


Figure 5. Scrollbar, Text Widget, and Corresponding IRG.

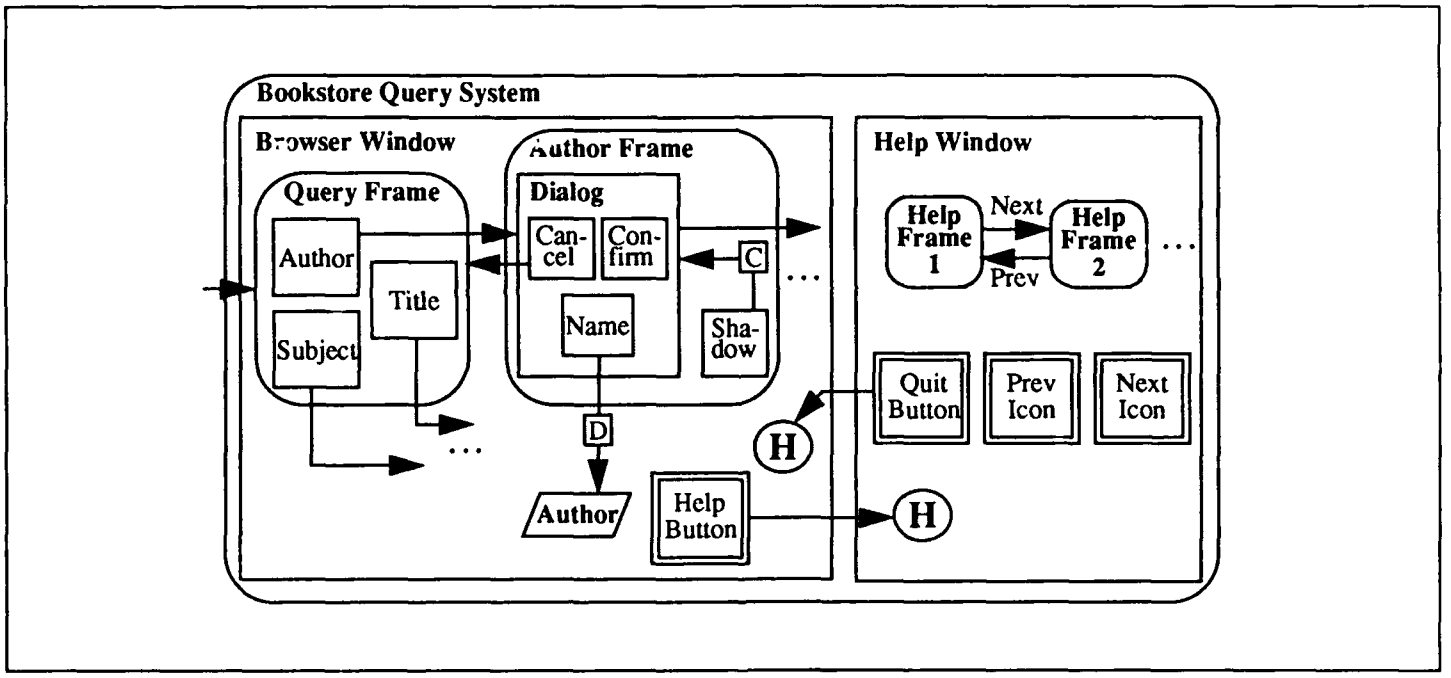


Figure 6. IRG Representation of Interface in Figure 3.

correspond to the first two frames in Figure 3. The four nodes in the Book node and the three nodes in the Author node represent the widgets in the frames that receive input. The Name node in the Author node does not have a transition leaving it because it represents a label widget and input to that widget does not cause any changes in the interface. The circles with the H in the middle are history states that return control to the last executed node. If no node has been executed, the default node is used. The transition from the Query node to the history node represents an input that can be executed anytime to take the user into the Help window.

VI. CONCLUSION

The system described here allows a wide range of user interfaces to be easily and quickly prototyped by an interface designer with little and in some cases no assistance from an applications programmer. It allows the interface components, their relationships, flow of control, constraints, and semantic feedback to be interactively laid out directly on the screen. The designer specifies the interface by defining windows, frames, subframes, and the graphics within them. The frames and graphics can then be grouped together for

easy specification of persistent objects and objects with identical attributes or flow of control. The grouping supports consistency of the objects' appearance and operation across the interface and reduces the time to modify persistent objects.

The IRGs are capable of representing the structure, flow of control, constraints, and semantic feedback of the interface and provides a formal specification that complements the prototype. The hierarchical structure of the IRGs allows them to represent the nested components of the interface and the grouping of the objects and their flow of control. It also has constructs for representing the execution of semantic actions of the application, feedback from the application, and constraints between graphics and the interface.

REFERENCES

1. Bass, L., Hardy, E.J., Hoyt, K., Little, R., and Seacord, R.C. (March 1988). *Introduction to the Serpent User Interface Management System.* Technical Report CMU/SEI-88-TR-5, ADA200085, Carnegie Mellon University, Software Engineering Institute.

2. Boehm, B. (October 1976). "Software Engineering," Software Series. TRW-SS-7-08, TRW Defense & Space Systems Group.

3. Boehm, B. (May 1984). "Prototyping Versus Specifying: A Multiproject Experiment," *IEEE Transactions on Software Engineering*, SE-10 (3), 209-302.

4. Cardelli, L. (October 1988). "Building User Interfaces by Direct Manipulation," *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, Banff, Alberta, Canada, 152-166.

5. Goodman, D. (1987). *The Complete HyperCard Handbook*, Bantam Books, New York.

6. Green, M. (1985). "Report on Dialogue Specification Tools," *User Interface Management Systems*, edited by Gunther E. Pfaff, Springer-Verlag, 9-20.

7. Harel, D. (May 1989). "On Visual Formalisms," *Communications of the ACM*, 514-528.

8. Interface Builder (1989), *NeXT Systems Reference Manual*, Chapter 8, NeXT Inc.

9. Lewis, T., Handlooser III, F., Bose, S., and Yang, S. (1989). "Prototypes from Standard User Interface Management Systems," *IEEE Computer*, 51-60.

10. Mayer, N., Shepherd A., and Kuchinsky, A., (May 1990). "Winterp: An Object-Oriented, Rapid Prototyping, Development Environment for Building Extensible Applications with the OSF/MOTIF UI Toolkit," *Xhibition '90 Conference Proceedings*, San Jose, CA, 49-64.

11. *MOTIF Programmer's Guide, Open Software Foundation* (1989), Open Systems Foundation, Cambridge, MA.

12. Musciano, C. (1988). *Tooltool Reference Manual.*

13. Myers, B. (1988). *Creating User Interfaces by Demonstration*, Academic Press, Boston, MA.

14. Myers, B., Guise, D., Dannenberg, R., Vander Zanden, B., Kosbiew, D., Marchal, P., Pervin, E., and Kolojejchick, J. (November 1989). *The Garnet Toolkit Reference Manuals: Support for Highly-Interactive, Graphical User Interfaces in LISP*, Technical Report CMU-CS-89-196, Carnegie Mellon University, Computer Science Department.

15. *Prototyper Reference Manual* (1987), SmethersBarnes.

16. Scheifler, R. and Gettys, J. (April 1986). "The X Window System," *ACM Transactions on Graphics*, 79-109.

17. Wellner, P. (April 1989). "Statemaster: A UIMS Based on Statecharts for Prototyping and Target Implementation," *Proceedings of the ACM SIG CHI '89*, 177-182.

18. Wilson, J. and Rosenberg, D. (1988). "Rapid Prototyping for User I terface Design," *Handbook of Human-Computer Interaction*, edited by Martin Helander, North-Holland.

19. *XBuild User's Guide* (1990), Nixdorf Corp, Cambridge, MA.