Input/Output Linkage in a User Interface Management System

Dan R. Olsen Jr.
Brigham Young University

Elizabeth P. Dempsey
Arizona State University

Roy Rogge
Arizona State University

Abstract

The GRaphical INteraction System (GRINS) is described, which integrates an automaton-based dialogue controller with a dynamic display model to provide a User Interface Management System. The linkage between the logical device interface and the graphical presentation of virtual devices is discussed. A display manager to support dynamic manipulations of hierarchically structure images is presented. Lastly a model of Display Objects whereby application-specific display objects can have computational constraints defined is described. The constraint system is equivalent to an attributed grammar and is evaluated using an incremental attribute flow algorithm.

INTRODUCTION

With the huge increase in the number of interactive graphics systems that are being programmed has come a desire to reduce the cost of building such systems and to increase the quality of their user interfaces. In an attempt to meet this need, there have been a number of systems developed or proposed which can be categorized as User Interface Management Systems (UIMS). The purpose of such systems is to provide programming tools to aid in the development of quality interactive systems. In most UIMSs that have been developed, the emphasis has been on organizing and translating interactive inputs into the actual program behavior that the user desires. Processing an interactive dialogue, however, requires both understanding the inputs and providing visual feedback about the input. The visual feedback portion of an interactive dialogue has received significantly less attention in UIMS research. This paper will discuss the GRaphical INteraction System (GRINS) which is a prototype UIMS which has been built to study this linkage between input language parsing and graphical feedback.

There is a variety of ways in which visual feedback is addressed in various UIMSs. Some UIMSs are primarily concerned with input parsing and provide no direct support for visual feedback. The transition networks of Jacob [Jaco 83] fall into this category. The input-output tools of van den Bos [vand 83] and the abstract devices of Anson [Anso 82] have a notion of encapsulating input/feedback concepts in a tool but do not discuss mechanisms for actually providing the visual feedback. Some UIMSs have builtin or static feedback techniques. These systems are dominated by their input language specifications and use a fixed set of primitive virtual devices whose feedback techniques are hardcoded into them. The menu trees of TIGER [Kasi 82] and the menus and simulated valuators of SYNGRAPH [Olse 83] have this characteristic. In these systems the UIMS provides a predefined external view or presentation for the interactive dialogue which cannot be significantly altered. There are systems such as Menulay [Buxt 83] and Flair [Wong 82] which provide tools for designing the presentation of a dialogue but the linkage between the presentation and the dialogue is not a strong one. This problem of input/feedback linkage became apparent in the design of the input primitives for the CORE [GSPC 79] and GKS [GKS 84]. Rosenthal and others [Rose 82] have carefully defined the kinds of feedback that a primitive device should support but this has not been integrated with a UIMS.

The paper will proceed by first discussing the overall GRINS architecture to identify where the input output linkages occur. This will be followed by a discussion of the lexical presentation issues including the layout editor which manipulates the lexical and syntactic level presentations of a user interface. The GRINS display manager with its parameterized templates and segments will be discussed as a model for dynamically manipulating images. Lastly the GRINS object description language for modeling application-specific display objects which provide the final level of input/output linkage will be presented.

GRINS ARCHITECTURE

The GRINS software consists of three components as shown in Figure 1. The first is a parser that parses a description of the interactive interface to be generated. The second is the layout editor which allows the programmer to graphically design the layout of the screens, menus, feedback space and prompts. The third component is the runtime system that interprets the information generated by the other two components and actually executes the interactive program.

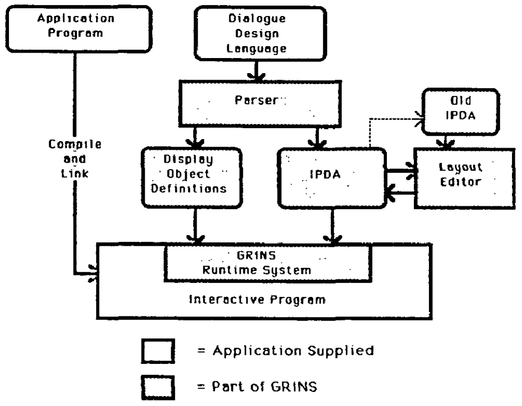

Figure 1.
Overall Structure of GRINS

The parser generates two files as its output. The first file contains the actual dialogue definition in the form of an Interactive Push-Down Automaton (IPDA). The definition of an IPDA and the algorithms for generating it from an input description are found in Olsen [Olse 84]. The Display Object Definitions file contains the information which describes the structure of the application.

The runtime architecture of GRINS is shown in Figure 2.

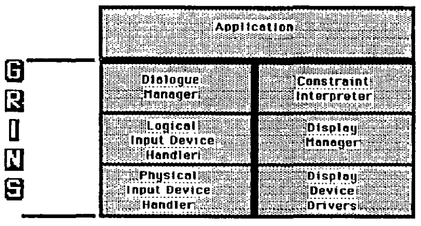

Figure 2.
Runtime Structure of GRINS

In addition to the application there are six components to the GRINS runtime system itself. There are three levels of input processing and three for output. The role of the Dialogue Manager is to interpret the IPDA and control the entire interaction. It does this by receiving inputs via the Logical and Physical Input Device Handlers and by making calls on the application and the Constraint Interpreter. In Figure 2 each component can communicate directly with those components that are adjacent to it. The primary emphasis of this paper is on the Display Manager and the Constraint Interpreter. We will discuss in detail how they interrelate with the Dialogue Manager and the Logical Input Device Handler. We will also discuss the models for dynamic graphical output that these components support and how they are specified.

LEXICAL PRESENTATIONS

In discussing how the presentation of an interactive dialogue is handled on the lexical level we must first understand how the Dialogue Manager interfaces with the Logical Input Device Handler. This will identify specific properties of logical input devices which must be reflected in the presentation. We will then discuss how this information is manipulated by the Layout Editor.

The Dialogue Manager/Logical Input Device Interface

At the lexical level, the primary presentation issues of a user interface are how the logical input devices are prompted for and how their inputs are echoed. The lexical level presentations are defined by the Layout Editor, are controlled by the Dialogue Manager and the Logical Input Device Handler and are actually displayed by the Display Manager.

The Dialogue Manager views logical input devices as being either event or sampled devices and as returning 0 or more typed attributes as their result. An obvious example is a tablet which is sampled and returns two real attributes. The interface between the Dialogue Manager and the Logical Input Device Handler makes no further distinctions between devices and has no preset enumeration of input devices.

Our present Logical Input Device Handler supports two classes of virtual devices in addition to the normal physical devices which are simply passed through the logical level. These virtual devices are menu items and picks. As in SYNGRAPH, picking is based on the type of the data object that a picked image represents. A new logical device class is created for each data type that can be picked and the attribute returned by each such device is the data value associated with the picked image. The menu items, however, are more relevant to the current discussion because of their close relationship with the presentation of an interaction. Menu items best illustrate the interplay of the interactive dialogue and the presentation.

In discussing the relationship between the Dialogue Manager and the logical input devices we will use the terms and definitions set forth for GKS by Rosenthal [Rose 82]. The Dialogue Manager accepts logical inputs by either consuming them, as in the case of event devices, or sampling them, as in the case of sampled devices. Because most interactive systems require more logical inputs than physically exist, the physical resources must be multiplexed to possibly more than one logical device. To ensure that there is no conflicting usage of physical resources, the IPDA parser partitions the dialogue into modes. Each mode is characterized by a set of logical devices which are accessible in that mode. When the Dialogue Manager enters a new mode it releases and acquires logical devices so that the set of logical devices that are actually accessible matches the set of devices defined for the new mode. If a physical resource is bound to two logical devices there is no conflict as long as those logical devices never appear in the same mode. The syntactic portion of the parse does not know about the bindings of multiple logical devices to a single physical device. It is the acquire and releasing process associated with a mode change which resolves the ambiguities.

Simply because a device is accessible does not mean that its input is acceptable at any point in time. The set of acceptable inputs is determined by the current state of the dialogue. When the dialogue manager changes states it disables all input devices that are no longer acceptable and it enables any that have become acceptable.

When an input is actually accepted by the Dialogue Manager it must be acknowledged. Because inputs usually are grouped into commands or possibly smaller "chunks" [Buxt 83] inputs should stay in an acknowledged state until closure is reached (as determined by the Dialogue Manager).

The Layout Editor

The role of the layout editor is to take the mode information provided in the IPDA and append additional presentation information to it. The Layout Editor integrates the principles presented in MENULAY [Buxt 83] with the dialogue control strategies developed from SYNGRAPH.

The editor constructs a layout for each mode in the IPDA. The general graphical presentation of the mode including titles, background colors, boundaries of viewports and menu areas and location of help and feedback message areas are all expressed graphically on the screen using the editor. In addition, the layout editor allows for the positioning of the menu icons for each logical menu device that is accessed by that mode.

In addition to the placement of menu icons, a facility for designing and editing icons is provided. Since each menu item represents a logical input device it must respond to the enable, disable, acknowledge and unacknowledge requests from the Dialogue Manager. This means that a logical device must present to an interactive user one of four states (enabled/acknowledged, enabled/unacknowledged, disabled/acknowledged and disabled/acknowledged). As an icon is drawn on the screen the primitives which make up the icon are divided into groups by the designer. Each of these groups can be edited separately and the designer can create as many of these groups as desired. (More than two or three groups is usually excessive.)

The purpose of groups is to allow for the presentation of the four states of an icon. Each icon has a color matrix like that shown in Figure 3.

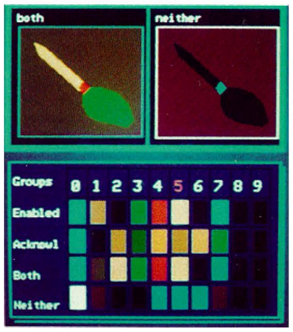

Figure 3.
Icon Color Matrix

As the figure shows, the rows of the matrix correspond to the states of the icon and the columns are the groups. Colors are selected from the palette to fill in the cells of the matrix. As an icon changes state the Logical Input Device Handler modifies the color of the icon's groups to match the settings of the appropriate row in the matrix. To aid the designer in creating these dynamic icons there is a drawing area for each of the four device states. Drawing in any of the areas updates the other three, using the appropriate color settings for each. By using the color matrix from previously designed icons one can easily and quickly design a uniform presentation for an entire dialogue.

We have only discussed the presentation of iconic menu items because they best illustrate the dialogue/presentation relationship. Other virtual devices are easily added to the Logical Input Device Handler.

DISPLAY MANAGEMENT

Our original intent in developing GRINS was to use CORE or GKS as a basis for the display portion of the system. Both of these were found to be inadequate, however, because of the inability to edit display primitives within segments. This editing ability is essential to being able to perform the rapid screen updates required in an interactive application. In addition, neither the CORE nor GKS has the geometric modeling capabilities that we desired to support display objects.

It is very important that the model for dynamic image modification be incorporated into the display manager because of the disparity of screen update techniques required by various display devices. If there is a basic model for image update then the display manager can be tailored to the needs and limitations of the display device whether it be vector refresh or raster.

Primitives, Segments and Templates

The basic graphical units in the display manager are primitives, templates, and segments. The primitives currently supported are lines. arcs, polygons and text. Primitives are grouped into templates in much the same fashion that they are placed in segments in CORE or GKS. The difference is that the arguments to each of the primitives can either be constant values or references to a segment parameter. Templates also contain transformation information for references to subsegments. The actual subsegment referenced, however, lies in the segment using the template. Each segment references a template which contains the primitives which describe how the segment should be graphically interpreted. The parameters can be real or integer numbers or character strings. Editing operations on this structure consist of either exchanging or replacing subsegments of a segment or changing a parameter value. A template is intended to be used by more than one segment each of which contains different editable values. The role of templates is to contain the relatively static portions of an image. A segment on the other hand contains the modifiable portion. When a primitive with a parameter is added to a template, an initial value for the parameter is also given. These initial values are stored in the template and are used when creating new segments for the template.

In addition to the simple primitives, segments and templates the display manager provides viewports and subsegment references. Subsegment references simply provide a hierarchical structure for pictures. With each subsegment reference there may or may not be a transformation matrix. The elements of the transformation matrix may also be parameterized. A viewport definition can be placed in the template of either the root segment or another viewport. Viewports contain their own viewing definitions which can be parameterized in the segments which use the viewport's template. Thus the editing model can be consistently applied to viewports as well as any other graphical object. Viewport templates can themselves contain primitives, other viewports or subsegments references.

This model for parameterizing display lists has existed for quite some time. Many vector refresh displays provide this feature in hardware. Similar structures are proposed by Michner [Mich 78]. Turner [Turn 84] has proposed extensions to GKS which are similar. The difference is that in Turner's model the graphics variables are global to the entire image whereas in our model the variables lie in the individual segments. This is a very important difference because a flat or unstructured variable space is inconsistent with a hierarchically structured image.

EXAMPLE: Menu Icons
Using the menu icons described previously as an example, each icon is represented by a template. For each group in the icon there is a color parameter stored in a segment. Each primitive added to the template uses the color parameter which is appropriate for its group. As a menu device changes state the Logical Input Device Handler needs only to change the parameters in the icon's segment to update the menu display. It is up to the display manager to perform the appropriate screen update operations.

EXAMPLE: Layouts
Using templates/segments for display management each mode's layout becomes a viewport template. This mode template contains the various viewports for the application's use and the menus. In addition, all of the other static layout information is stored in this template. When the dialogue changes mode the Logical Input Device Handler simply changes the root segment/template.
There are several of these layout segment/template pairs, one for each mode. The only one that is ever displayed

is the one referenced by the root segment. All layout segments reference the same application world segment (which of course may itself have subsegments). Menu icon segments are referenced as subsegments of the layout. Each layout references that subset of the menu icon segments which are part of the mode that the layout represents. Figure 4 shows the structure of layouts, application viewports and menu icons using this model.

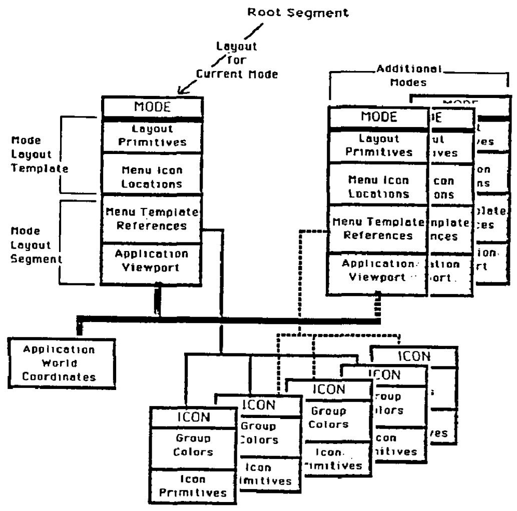

Figure 4.
Structure of Layouts, Viewports and Menu Icons

This model for a display manager provides a simple and straightforward method for dynamically manipulating images. For purposes of providing feedback from more dynamic input techniques than simple menu icons this model is not sufficient. As Turner has already shown, a computational linkage between input values and output parameters must be provided. The display objects to be described later provide this capability.

Picking

As a final note about the interface between the Logical Input Device Handler and the Display Manager, each template has stored in it a type code to identify the type of application data that it represents and each segment has stored in it a pointer to a data value. This information in conjuction with the types of the enabled logical pick devices is used by the Logical Input Device Handler to resolve pickability and picking ambiguity problems from the screen.

DISPLAY OBJECTS

In providing a dynamic output structure whose power is consistent with our dialogue control system we wanted to be able to model a wide variety of interactive techniques. We were guided in this by several principles. The first, which has already been stated, is that simple value substitution does not have sufficient power to link inputs to outputs. The second is that the input/output linkage is frequently application specific. The third principle is that an image represents an application data object whose presentation is being defined.

In the Dialogue Design Language the programmer provides what are called display object definitions. Display objects contain the computational portion of the output definition. As a simple example of a display object we define a virtual valuator in the form of a slider as shown in Figure 5.

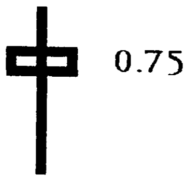

Figure 5.
A Virtual Valuator.

By picking a location on the slider shaft we want to move the slider up and down. We also want the application to be able to set the upper and lower bounds of the slider and to be able to read the current value of the slider. Figure 6. shows the object definition of the slider.

```
Object Slider( Max,Min:Real;
 ! Max and Min define the range for the
 ! slider in the units of the application
    SliderVal:Real
 ! This returns the slider value
    ) =
 ! This display object is defined in a
 ! local coordinate system ranging from
 ! 0.0 to 1.0.
Control YLoc: Real := 0.5;
 ! This is the slider location which is
 ! set by the dialogue manager.  This is
 ! defined in the coordinates of the
 ! slider object.
Assert (YLoc <= 1.0) and (YLoc >= 0.0);
Def SliderVal:=YLoc*(Max-Min)+Min;
 ! This is the value of the slider which
 ! the application can read.
Line( (0.0,1.0),(0.0,0.0),Black);
 ! This is the slider range
Def SliderTop:=YLoc+0.1;
    SliderBottom:=YLoc-0.1;
Polygon( (0.1,SliderTop),(-0.1,SliderTop),
   (-0.1,SliderBottom),(0.1,SliderBottom),
    Black);
Text( RealToString(SliderVal),
(0.11,YLoc),0.0,0.1, Black);
EndObject;
```

Figure 6.
Slider Object Definition.

Note that several of the display parameters are indirectly computed from the value YLoc. This value can be changed by the Dialogue Manager. This value could also be changed by the application if desired. After a control value or set of values is changed all assertions are then checked. If any assertion fails then the change is ignored and the image remains unchanged. If all assertions are valid then the all of the implied values which are found in Def statements are computed including the actual SliderVal.

The Line, Polygon and Text statements define the graphical primitives of the actual objects. Note that some of their arguments are constants while some are computed from the control variables and object parameters. For each object definition a template is created in the Display Manager. All of the object's graphical primitives are placed in the template with those arguments that must be computed being defined as parameters. After the implied values are computed the corresponding segment parameters that have changed are updated and the Display Manager updates the screen.

This model provides a nice separation of tasks. The application sees only Max, Min and SliderVal in its own units. The Dialogue Manager sees only YLoc and needs only know the coordinate system of the slider. The presentation of the slider is independent of all of the other components and can be changed as needed.

The slider object can be used to form a 3D locator object as shown in Figure 7.

```
Object Loc3D( CoordMax, CoordMin: Real;
      LocX,LocY,LocZ :Real ) =

  S     u     b     O     b     j
X:Slider(CoordMax,CoordMin,LocX);
    Y:Slider(CoordMax,CoordMin,LocY);
    Z:Slider(CoordMax,CoordMin,LocZ);
EndObject;
```

Figure 7.
3D Locator Object Definition

Note that the values of Max and Min for each slider are propagated down from the 3DLocator object and that their SliderVal value are propagated up as LocX, LocY and LocZ. This propagation of values is functionally identical to an attributed grammar. The algorithm used to determine which values (or attributes) must be recomputed after a change is the incremental attribute flow algorithm of Demers [Deme 81].

Note also that the functions which compute the assertions and the implied values can be application-specific functions. This meets the requirement defined at the Seattle Workshop on Graphical Input and Interaction Techniques [Thom 83] that any mechanism allowing a UIMS to change the form of a displayed image must have a provision for the application to veto the change.

SUMMARY

The GRINS user interface management system has addressed the issues of how models for dynamic displays can be integrated with dialogue control to provide a comprehensive set of principles covering both the input and output facets of an interactive dialogue. GRINS has been implemented in Pascal and C under UNIX and using a Raster Technologies display. We are pleased with the progress that we have made with this system, however, we feel that additional work needs to be done on the screen update algorithms of the Display Manager to improve its efficiency and we are hoping to gain much more experience with the power and applicability of our Display Objects.

REFERENCES

Anson, Ed. "The Device Model of Interaction." Computer Graphics 16, 3 (July 1982) pp. 107-114.

Buxton, William. "Lexical and Pragmatic Considerations of Input Structures." Computer Graphics 17, 1 (January 1983) pp. 31-37.

Buxton, W, Lamb, M.R., Sherman, D., Smith, K.C. "Towards a Comprehensive User Interface Management System." Computer Graphics 17, 3 (July 1983) pp. 35-42.

Demers, A., Reps, T. and Teitelbaum, T. "Incremental Evaluation for Attribute Grammars with Application to Syntax directed Editors." 8th Conference on Principles of Programming Languages (January 1981) pp. 105-116.

Graphical Kernel System, ANSI X3H3/83-25r3; Special Issue, Computer Graphics (February 1984).

GSPC. "Status Report of the Graphics Standards Planning Committee." Computer Graphics 13, 3 (Aug 1979).

Jacob, R.J.K. "Using Formal Specifications in the Design of a Human-Computer Interface." Communications of the ACM 26, 4 (April 1983).

Kasik, David J. "A User Interface Management System." Computer Graphics 16, 3 (July 1982) pp. 99-106.

Michner, J.C. "A Graphics System for Real-Time Programming." Proceedings of the Society for Information Display, Vol 19, 4 (Fourth Quarter 1978) pp. 157-161.

Olsen, Dan R. and Dempsey, Elizabeth P. "SYNGRAPH: A Graphic User Interface Generator." Computer Graphics 17, 3 (July 1983) pp. 43-50.

Olsen, Dan R. "Push-down Automata for User Interface Management." ACM Transactions on Graphics 3, 4 (July 1984).

Rosenthal, D.S.H, Michener, J.C., Pfaff, G., Kessener, R. and Sabin, M. "The Detailed Semantics of Graphics Input Devices." Computer Graphics 16, 3 (July 1982) 33-38.

Thomas, James J. and Hamlin, Griffith. "Graphical Input Interaction Technique: Workshop Summary." Computer Graphics 17, 1 (January 1983) pp. 5-30.

Turner, Joshua U. "A Programmer's Interface to Graphics Dynamics." Computer Graphics 18, 3 (July 1984) pp. 263-270.

van den Bos, J., Plasmeijer, M.J. and Hartel, P.H. "Input-Output Tools: A Language Facility for Interactive and Real-Time Systems." IEEE Transactions on Software Engineering SE-9, 3 (May 1983) pp. 247-259.

Wong, Peter C.S. and Reid, Eric R. "Flair - User Interface Dialog Design Tool." Computer Graphics 16, 3 (July 1982) pp. 87-98.