AUTOMATIC GENERATION OF GRAPHICAL USER INTERFACES

Gurminder Singh and Mark Green

Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1

ABSTRACT

The research reported here is focussed on the issues involved in automatically generating the presentation component of user interfaces. The design and implementation of the presentation component of the University of Alberta User Interface Management System are described. The system is used for automatically generating graphical user interfaces for interactive applications. The system has been designed to keep the other components of the user interface device independent, keep the designer's interest alive in the design process, make the design process less cumbersome, and reduce the burden of programming as far as possible. The results presented in this report are based on the experience gained through implementing a system to generate the presentation component of user interfaces automatically. The presentation component can be viewed as the lexical level of the user interface.

1. Introduction

There has been a growing awareness in software design of the importance of the user. This concern has manifested itself, for example, in analysis of desirable properties of user interfaces [Cheriton76] and in investigations into the user-friendliness of interactive systems. The concept that the user interface can be treated as a separate module within the whole system, and not simply embedded at a range of points through it, is gaining acceptability [Edmonds81]. The effort now is to make user interfaces more interactive, graphic, forgiving, and self-explanatory. But, unfortunately, the construction of good user interfaces even today remains an expensive, time-consuming, and often a frustrating process [Buxton83]. This prompted researchers in human factors to explore the possibility of automatically generating user interfaces and the notion of a User Interface Management System (UIMS). This paper describes a tool for automatically generating graphical user interfaces for interactive programs and explores the issues related to the process.

1.1. What is A User Interface?

The user interface is the part of a system that handles the interaction between the user and the other components of the system. In order to complete a useful task the system accepts inputs and presents outputs through the user interface. As more interactive systems of comparable functionality become available, their success in the market place is based increasingly on ease of use. Bad user interfaces often cause unnecessary loss of productivity and aggravation. Ease of use, not ease of implementation, has become the crucial design consideration.

The basic structure of a user interface does not change radically over a wide range of applications [Green84a]. There are a number of functions that must be performed by most user interfaces. These functions include error detection and recovery, user protocoling, and undo processing. The concepts of a separate user interface module, separate interface designer, and the common features of the user interfaces have lead to the notion of UIMS.

1.2. Automatic Generation of User Interfaces

The fact that the basic structure of a user interface does not change radically over a wide range of programs and that functions like error detection, error recovery, and help are common to almost all user interfaces leads to the idea of automatic generation of user interfaces. The high cost and large turnaround time for hand coded user interfaces provides additional motivation for the idea.

The automatic generation of the user interfaces has the following advantages:

1) It reduces the cost of producing user interfaces.

2) It provides a much shorter lead time than the hand coding of the interfaces.

3) The low cost and short lead time for the production of the user interfaces makes it possible to experiment with new ideas in user interface design.

4) Once the user interface generator is debugged completely, the software it generates is more reliable than hand coded software.

5) A particular user interface generator may be used to generate a number of user interfaces which are consistent in their over all approach to functions such as error reporting and help. Familiarity with one such user interface can expedite the learning of the others.

1.3. What is a UIMS?

A UIMS is a collection of software tools supporting the design, specification, implementation, and evaluation of user interfaces [Seattle83]. It performs an important role of mediating the interaction between a user and an application; satisfying user requests for application actions, and application requests for data from the user. It thus provides for the application programmer's problem specific skills to be concentrated on the application, and freed from detailed concern with managing the flow of user actions and responses. UIMSs have also been called "Dialogue Management Systems" [Roach82] or "Abstract Interaction Handlers" [Feldman82]. Over the past few years many models of UIMSs have been proposed and implemented [Newman68], [Kasik82], [Guest82], [Buxton83], [Jacob83], [Olsen Jr.83].

2. The University of Alberta UIMS

The University of Alberta UIMS [Green85], [Singh85], [Lau85], [Chia85] is based on the Seeheim model of user interfaces discussed in section 2.1. The design and implementation details of the presentation component of the U of A UIMS are described in this paper. Three main notations have been used for specifying the dialogue between the user and computer. These notations are: recursive transition networks, BNF grammars, and events. A system accepting dialogues specified by recursive transition networks is discussed in [Lau85]. Details about an event language and its implementation can be found in [Chia85]. At the present time the implementation of a grammar based notation has not been started. Support for the application interface model is currently under development.

2.1. The Seeheim Model of User Interfaces

In the Seeheim model of user interfaces [Green84b] a user interface is divided into three components as shown in Figure 1. The presentation component can be viewed as the lexical level of the user interface. This component is responsible for managing the input and output devices used by the user interface. All the interaction techniques and display formats are defined in this component. It reads the physical input devices and converts the raw input data into the form required by the other components in the user interface. The user interface employs an abstract representation for the input and output data. This representation consists of a type or name that identifies the kind of data, and the collection of values that define the data item. This chunk of information is called a token.

While the presentation component is responsible for converting user actions to input tokens, the dialogue control component defines the set of legal input tokens. It interprets the sequence of input tokens produced by the presentation component to determine the operations the user wants to perform. Once a complete command has been formed from the input tokens the dialogue control component uses the application interface model to invoke the appropriate routines in the application. Similarly the output tokens sent by the application interface model are interpreted by dialogue control and transformed into a format acceptable to the user. This component contains the control logic of the user interface. Most existing UIMSs have concentrated on this component of the user interface.

The application interface model is a representation of the functionality of the application. It represents the user interface's view of the application. The application interface model contains the descriptions of the major data structures maintained by the application, and the application routines that can be invoked by the user interface. It also covers the mode of communication between the user interface and the application. The user interface may communicate with the application in one of the three possible modes of interaction: the user initiated, the system initiated, and mixed initiative.

3. Design of the Presentation Component

The basic job of the presentation component is to convert user interactions with the input devices into input tokens, and convert output tokens into images on the output devices. The basic structure of the presentation component of the University of Alberta UIMS is presented in Figure 2. In the following sections a description of the important concepts related to the presentation component of the University of Alberta UIMS is presented.

3.1. Input Tokens

The input tokens convey information about the user's interactions with the user interface to the other parts of the UIMS. The raw data generated by the user interactions with the input devices is manipulated and restructured by the interaction techniques and control module. This new structure, called an input token, is sent to the other parts of the UIMS. An input token represents exactly one unit of information as far as the UIMS is concerned. An input token contains the following information.
- Token Number
- Token Value

The token number is the unique number assigned to each type of input token by the presentation component of the UIMS. The interpretation of the token value depends upon the token number.

3.2. Output Tokens

The output tokens are used for generating images. These tokens are generated by the dialogue control component as well as the application, and are sent to the control module of the presentation component for further processing. The control module invokes the display procedure associated with the output token and ensures that the image is generated in the appropriate window. An output token contains the following fields.
- Token number
- Token value

The token number is the unique number assigned to each type of output token by the presentation component of the UIMS. Using the token number as the key the control module finds the associated display procedure and the window name. The interpretation of the token value is left to the display procedure. Usually this field points to a structure defined in the token definition file. The token definition file contains the definitions of the structures the value field of an input or output token could point to.

3.3. Control Module

The control module is responsible for all communication with the other parts of the user interface. This communication includes sending the input tokens to the dialogue control component and receiving the output tokens form the other parts of the user interface.

The other important function of the control module is to perform an external-internal mapping. This mapping determines how the user's actions are converted into input tokens and how output tokens are converted into images. The external-internal mapping can be viewed as a dictionary used by the control module for interpreting user actions and output tokens. For input tokens the control module uses the event number and the window name of the input event to determine the input token number. In the case of output tokens, it determines the window where the image will be generated and the display procedure to be used from the output token number.

3.4. Library of Interaction Techniques

Most interaction tasks are performed by a set of interaction techniques. An interaction technique is defined as a way of using a physical input device to enter a certain type of word (command, value, location, etc.), coupled with the simplest form of feedback from the system to the user [Foley81].

There are a large number of possible interaction techniques. Each interaction technique is suitable for a particular function. The set of interaction techniques available to a designer remains very limited if he/she has to develop one every time it is required. To make the designer's choice wider a library of interaction techniques can be created. This library can be used by any designer while deciding on which interaction techniques to use. Every time a new technique is developed it can be added to the already existing library. In this way one can keep building the library and help reduce the cost and time of producing good user interfaces.

3.5. Library of Display Procedures

A display procedure is a procedure that consumes output tokens. In the process of consuming output tokens the display procedure produces images on the graphics display. This image represents the data received in the output token.
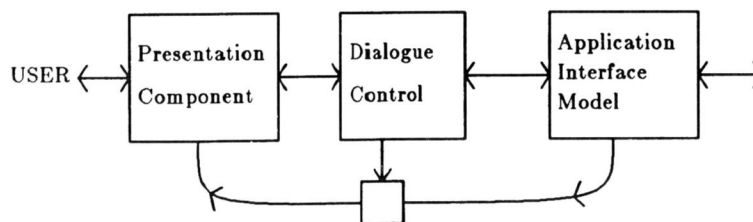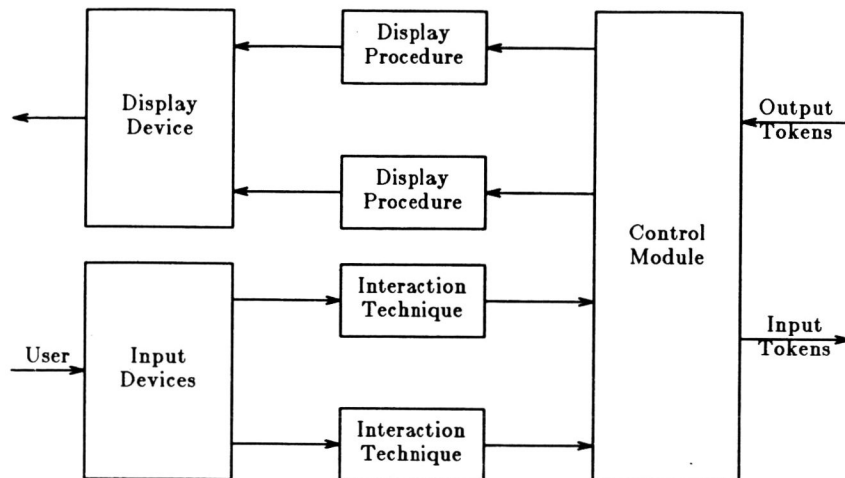

Fig. 1. The Seeheim Model of a User Interface

Fig. 2. The Structure of the Presentation Component

Each display procedure has a specific purpose and is used to generate a specific image. Examples of display procedures are Angle Display, Vertical Bar Display, and Text Windows.

The idea of having a library of display procedures is similar to that of the library of interaction techniques. By adding new display procedures to the library a large body of procedures can be built. This library can then be used for fast and economical production of user interfaces.

4. The Implementation

The presentation component of the University of Alberta UIMS has been implemented on VAX 11/780 running UNIX* 4.2 BSD. The programs used for implementing the system are written in the programming language C.

4.1. Structure of the Presentation Component

The presentation component of the University of Alberta UIMS is responsible for the following activities.

- Screen management
- Information display
- Associating interaction techniques with windows
- Associating display procedures with output tokens
- Assigning unique token numbers to input and output tokens
- Converting user interactions into input tokens
- Converting output tokens into images
- Lexical feedback
- Adapting to different display devices, if possible

An interactive approach to the design of the presentation component of the user interfaces has been adopted in this UIMS. There are two steps involved in the complete design. The first step is the specification step. In this step the user interface designer interactively specifies the design information. This information is then used in the second step for generating the presentation component of the user interface and providing run-time support. To support both these functions the presentation component of the University of Alberta UIMS is divided into two logically independent parts. The first part, called "ipcs" (interactive presentation component specification), accepts the design specifications from the designer and generates a data base, token tables, and 'C' procedures. The second part, called "pcg" (presentation component generation), consists of a number of procedures which provide run-time support for the presentation component of the user interfaces. This part is driven by the data base and the token tables. The 'C' procedures produced by ipcs are compiled and linked with the pcg procedures. The entire sequence of creating a presentation component is

shown in Figure 3. In implementing the presentation component a graphics package called WINDLIB [Green84c] and a data base package called FDB [Green83] are extensively used.

4.2. The Specification Step

The complete specification of the presentation component involves the specification of the following components.

- Screen layout
- Input tokens and interaction techniques associated with windows
- Output tokens and display procedures
- Menu layouts

The interactive specification program "ipcs" is used to enter the design information. The ipcs screen is divided into four areas as shown in Figure 4. The work area corresponds to the display screen of the user interface being designed. The designer positions windows and menus in this area. Above the work area is a text area used for help and error messages. The right side of the screen is used for the ipcs menu. An area across the bottom of the screen displays some of the attributes of the current window in the work area. In the following sections a brief description of the functions performed by ipcs is presented.

4.2.1. Window Definition

Ipcs starts off by displaying the layout shown in Figure 4. At the start of the specification session the work area corresponds to the device window on which the user interface will be implemented. All the windows created by the user interface designer are children of this window. To start defining windows the designer selects the "Window Definition" command from the ipcs menu. A window can then be defined by pointing at its two opposing corners. Once a window is defined it can be removed, stretched to a different size, or moved to a new position. An arbitrary number of windows can be defined at each level.

4.2.2. Window Attributes

The window attributes can be defined by selecting the "Window Attributes" command from the ipcs menu. This command assigns and displays default attributes for each window in the work area. The default attributes consist of the window name, window limits, background colour, drawing colour, and boundary colour. The value of an attribute can be changed by pointing at it and entering a new value.

In this step an interaction technique can be associated with a window. An interaction technique can be selected from the library of interaction techniques, or it can be a procedure written by the user interface designer. The name of the output token associated with the window is also specified in this step. On receiving this token the run-time support

module creates the window. It is important to note that it is the output token name, not the window name, which is used by the run-time support module.

Ipcs is capable of handling windows of variable size. In a normal case, depending upon the size of the window, one, five, or ten window attributes are displayed at a time. In the case of one or five attributes, the next set of attributes can be displayed by moving the tracking cross inside the window and hitting carriage return. To be able to define or change attributes for a very small window, its size can be temporarily adjusted. The size of such a window is adjusted for the purposes of display only.

The system also allows the use of overlapping windows. In the case of overlapping windows, the attributes displayed in one window overlap with the attributes in the other window. These windows can be flipped by pointing at the desired window. The selected window temporarily becomes the top-most window and its attributes become visible.

It is not necessary to complete the definition of all the attributes of a window at one time. The designer can postpone the definition of all or some of the window attributes for a later time. The system does not force any predetermined sequence of specification steps to be followed.

The attributes for a window can be changed as often as desired. The system does not differentiate between changing a default attribute, or a designer-defined attribute. This facilitates the interactive design of user interfaces. This mechanism for accommodating changes in the specification also helps in adapting user interfaces to individual users. The interaction technique or colours associated with a window, for example, can be easily changed to the actual user's liking.

4.2.3. Menu Definition

The "Menu Definition" command is used for defining menus. A menu is always associated with one of the windows defined in the work area. A menu consists of a menu header and a variable number of menu items. The menu header contains information affecting the appearance and location of the menu. This information consists of menu name, menu type, menu items placement option, menu orientation, and menu output token. Menu name is entered by the designer. A meaningful menu name can be used to remember the purpose or contents of the menu. The system provides facilities for fixed as well as pop-up menus. The default menu type is "fixed". The placement of menu items can be automatically handled by the system at the run time. The menu area is equally divided and allocated to each menu item in the menu. The system then centers the text and icons. The designer, however, has the option of specifying the location and size of individual menu items in the menu. The default menu items placement option is "system", it can be changed to "designer". The default menu orientation is "vertical", it can be changed to horizontal. Each menu is assigned an output

```
┌─────────────────────┐
│      Design         │
│   Specifications    │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│       ipcs          │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│ Genarate Data Base, │
│  Token Tables, and  │
│    'C' Procedures   │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│  Compile and Link   │
│    'C' Procedures   │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│     pcg ready       │
└─────────────────────┘
```

Fig. 3. Sequence for Constructing a Presentation Component

token. On receiving this output token the run-time support module displays the menu. The run-time support module recognizes the menus by their output token, not by the menu names.

A variable number of menu items can be associated with a menu. Each menu item occupies a rectangular area within the menu area. A menu item can be labeled either by an icon or text strings. A menu may consist of a mixture of iconic and textual menu items. For each menu item the designer specifies its type; iconic or textual. In the case of textual menu items one or more lines of text can be associated with the item.

In the case of icons the system provides a library of icons. An icon may be selected from this library, or it can be produced by using an interactive editor, called ICON [Green86], developed at the University of Alberta. To associate an icon with a menu item the name of the procedure drawing the icon is entered.

The system allows the designer to associate more than one menu with a window. This helps in creating a hierarchy of menus. It is important to note that the menus may be displayed in any order. It is not necessary to display the menus in the order they are defined. Therefore, though the menu definition hierarchy is simple, the menu display hierarchy can be as complex as desired.

4.2.4. Input Token Definition

Input tokens can be associated with a window by selecting the "Input Token Definition" command from the ipcs menu. The input tokens convey information about the user's interactions with the user interface to the other parts of the UIMS. A variable number of input tokens can be associated with a window. An input token definition consists of token name and the associated event number. The system allows the designer to delete or add any number of input tokens during a specification session.

4.2.5. Output Token Definition

Output tokens can be associated with a window by selecting the "Output Token Definition" command from the ipcs menu. For output tokens the designer specifies the name of the token along with the name of a display procedure. On receiving the output token, the run-time support module invokes the associated display procedure. Based on the information contained in the output token, the display procedure produces the image in the window in which the output token is specified.

More than one output token can be associated with a window. Different output tokens are used to produce different images in the same window. The system allows the designer to modify or delete an output token after its definition.

4.2.6. Next and Previous Level Definitions

To be able to define the next level in the tree of windows, the designer can select "Next Level" command from the ipcs menu and point to a window in the work area. This window now becomes the new parent. Some of the important attributes of this window are displayed in the bottom of the screen and the environment switches back to the one shown in Figure 4.

A tree of windows can be created by using the "Next Level", "Previous Level", and "Window Definition" commands from the ipcs menu. By selecting the "Next Level" command and pointing at a window in the work area, children of the window pointed at can be created. The work area corresponds to the selected window and child windows can be created using the "Window Definition" command.

The system does not require the designer to complete the definition of the current level before going to the next level in the tree of windows. The tree can be defined branch-by-branch, level-by-level or by a mixture of the two approaches. This flexibility in designing the tree allows the designer to work more methodically and concentrate on one part of the user interface at a time. The system does not put any limit on the depth of the tree or number of windows at a particular level of the tree.

4.3. Novice and Expert Users of ipcs

Ipcs has two levels of use; "novice" and "expert". A novice may use only the basic set of commands. In this mode
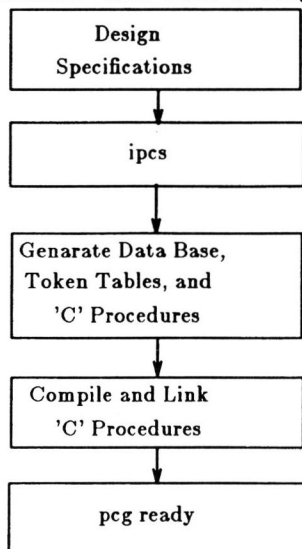
To change an attribute, move the cursor on top of the attribute and type the
new value. Hit carriage return after entering the value.
PF2 - to adjust size, PF3 - to flip windows, PF4 - to redraw

Window
Definition

Name: w9

Limit lbx: 0.00

Limit lly: 0.00

Limit urx: 1.00

Limit ury: 1.00

Bg Color: 0

Draw Color: 1

Bndry Color: 1

Intrct Tech.: none

Output Token: w9

Name: w7

Name: w8

Limit lbx: 0.00

Limit lly: 0.00

Limit urx: 1.00

Limit ury: 1.00

Bg Color: 0

Draw Color: 1

Bndry Color: 1

Intrct Tech.: none

Output Token: w8

Window
Definition

Window
Attributes

Menu
Definition

Input Token
Definition

Output Token
Definition

Next
Level

Previous
Level

EXIT

Help
ON OFF

Window	Parent	Bg Color	Draw Color	Boundary	Int. Tech.	In Tokens	Out Tokens
Jup7	Jup7	0	1	1	none	0	0

Fig. 4. The ipcs Screen Layout

all the input is done through one button on the cursor puck, and all through the ipcs session the designer is guided by help messages. The help messages are quite detailed for the novice. In expert mode, the designer is allowed to use other buttons on the puck, and the messages produced by the system are terse. The use of extra buttons on the puck allows the designer to delete, resize, move, or temporarily change the size or priority of a window. A profile for each user of ipcs (called "userprofile") is created and maintained by the program, initially tagging each user as a novice. The userprofile file stores the status ("novice" or "expert") of the user, the number of times he has invoked ipcs, and the number of times the designer successfully finished the specification sessions. A novice is upgraded to an expert based on the number of successful executions of the ipcs.

4.4. Output of ipcs

The output produced by ipcs is used to drive the run-time support module, and provides the interface for the dialogue control component and the application interface model. The information produced for the run-time support module consists of an FDB data base and 'C' procedures, whereas the interface information for the dialogue control component and application interface model consists of tables of input and output tokens. A brief description of these files is presented in the following sub-sections.

4.4.1. Data Base Description.

The design information for the presentation component is stored in an FDB data base. An object in the data base is represented by a frame. A window frame points to the next window at the same level as well as to its child windows at the next level. In addition, a window frame points to the first menu, first input token, and first output token frames associated with the window. Each of these in turn point to the frames providing detailed information about the objects.

4.4.2. 'C' Procedures

The 'C' procedures generated by ipcs are mainly used for passing parameters to the interaction techniques. The procedure calls are also used for loading the appropriate routines from the library of interaction techniques, display procedures, and icons. These procedures are compiled and linked with the other routines for run time support.

4.4.3. File of Input/Output Tokens

This file contains the names of all the input and output tokens along with a number. These numbers are assigned by ipcs to each of the input and output tokens. For reasons of efficiency this number is used for communication amongst various components of the user interface at run time.

4.5. Run-Time Support Module

The run-time support for the presentation component of the user interface is provided by the routines in "pcg". The following functions are supported by pcg.

- Receive the user's interactions in the form of WINDLIB events, reformat them into input tokens, and send these tokens for further processing.

- Receive output tokens from the dialogue control component, find the appropriate window and display procedure, and call the display procedure.

- Display menus and highlight the selected menu items.

The program pcg is driven by the data base created by the specification program ipcs. It retrieves the information from the database and restructures the information as required. It also receives some help from the 'C' code generated by the ipcs. This code is compiled and linked with the other run time routines. Pcg is divided into three logical parts. The first part is responsible for displaying menus and performing the associated bookkeeping. The second part handles the user interactions and generates the input tokens. The third part receives the output tokens and is responsible for their display.

5. Conclusions

In our system attention is focussed on the issues involved in the automatic generation of presentation components of user interfaces. The separation of presentation component from the dialogue control component helps designers work more methodically, and may therefore result in better user interfaces. The approach also overcomes one of the major stumbling blocks in user interface design, namely, the representation of geometric information in textual form. In our design most of the geometrical information is entered graphically. The system provides a window based environment which helps designers structure the user interfaces in a more natural way. The system also provides facilities for creating and maintaining a hierarchy of windows and menus. The interactive design of user interfaces is supported by allowing the designer to move to the next level in the hierarchy without completing the definition of all aspects of the user interface at the current level. The system provides more freedom to the designer by not imposing any predetermined sequence of commands for creating user interfaces.

The second contribution of this system is to show that all device dependencies can be limited to the presentation component of the user interface. If the user interface is moved to a different device only the presentation component needs to be changed. This increases the portability of the user interfaces. Also, the presentation component can be designed to support a range of devices, and automatically adapt to the one in use without changing the structure of the dialogue.

The third contribution of this system is to show that user interfaces can easily be adapted to individual users. Screen layout, for example, can be easily tailored for left handed users. The selection of interaction techniques and display formats can also be easily changed to the actual user's liking.

It is also observed that the existence of a separate presentation component encourages the use of a standard library of interaction techniques. This speeds up the process of generating user interfaces to a great extent and reduces the cost of programming considerably. This reduction in cost and time encourages experimentation with user interfaces and hence increases user satisfaction.

References

Buxton83.
W. Buxton, M. R. Lamb, D. Sherman and K. C. Smith, Towards a Comprehensive User Interface Management System, *Computer Graphics 17*, 3 (July 1983), 35-42.

Cheriton76.
D. R. Cheriton, Man-Machine Interface Design for Time-Sharing Systems, *Proc. of ACM Annual Conf.*, 1976, 362-366.

Chia85.
M. S. Chia, *An Event Based Dialogue Specification for Automatic Generation of User Interfaces*, M.Sc. Thesis, Dept. of Computing Science, Univ. of Alberta, Edmonton, Alberta, Canada, 1985.

Edmonds81.
E. A. Edmonds, Adaptive Man-Computer Interfaces, in *Computing Skills and the User Interfaces*, M. J. Coombs and J. L. Alty (ed.), Academic Press, London, 1981, 389-426.

Feldman82.
M. Feldman and G. Rogers, Towards the Design and Developement of Style Independent Interactive Systems, *Proc. 1st Annual Conf. on Human Factors in Computer Systems, Gaithersburg Maryland*, Mar. 1982, 111-116.

Foley81.
J. D. Foley, V. L. Wallace and P. Chan, *The Human Factors of Interaction Techniques*, IIST Report 81-03, Dept. of Electrical Engineering and Computer Science, The George Washington University, Washington, D.C., Mar. 1981.

Green83.
M. Green, M. Burnell, H. Vernjak and M. Vernjak, Experience with a Graphical Data Base System, *Proc. Graphics Interface'83*, 1983, 257-270.

Green84a.
M. Green, *The Design of Graphical User Interfaces*, Ph.D. Thesis, Univ. of Toronto, Toronto, Canada, 1984.

Green84b.
M. Green, Report on Dialogue Specification Tools, *Computer Graphics Forum 3*, (1984), 305-313.

Green84c.
M. Green and N. Bridgeman, *WINDLIB Programmer's Manual*, Dept. of Computing Science, Univ. of Alberta, Edmonton, Alberta, Canada, 1984.

Green85.
M. Green, The University of Alberta User Interface Management System, *Proc. Siggraph' 85*, 1985, 205-213.

Green86.
M. Green and G. Singh, Windows as a User Interface Structuring Mechanism, *(in preparation)*, 1986.

Guest82.
Stephen P. Guest, The Use of Software Tools for Dialogue Design, *Int. Journal of Man-Machine Studies 16*, (1982), 263-285.

Jacob83.
R. J. K. Jacob, Executable Specifications for a Human-Computer Interface, *Proc. CHI'83*, Dec. 1983, 28-34.

Kasik82.
D. J. Kasik, A User Interface Management System, *Computer Graphics 16*, 3 (July 1982), 99-106.

Lau85.
S. C. Lau, *The Use of Recursive Transition Networks for Dialogue in User Interfaces*, M.Sc. Thesis, Dept. of Computing Science, Univ. of Alberta, Edmonton, Canada, 1985.

Newman68.
W. M. Newman, A System for Interactive Graphical Programming, *Proc. Spring Joint Computer Conf.*, 1968, 47-54.

Olsen Jr.83.
Dan R. Olsen Jr. and Elizabeth P. Dempsey, SYNGRAPH: A Graphical User Interface Generator, *Computer Graphics 17*, 3 (July 1983), 43-50.

Roach82.
J. Roach, R. Hartson, R. Ehrich, T. Yunten and D. Johnson, DMS: A Comprehensive System for Managing Human-Computer Dialogue, *Proc. 1st Annual Conf. on Human Factors in Computer Systems, Gaithersburg Maryland*, Mar. 1982, 102-105.

Seattle83.
Seattle, Proc. of Graphics Input Interaction Technique Worksop, June 2-4 Battelle Seattle, *Computer Graphics*, Jan. 1983.

Singh85.
G. Singh, *Presentation Component for the U of A UIMS*, M.Sc. Thesis, Dept. of Computing Science, Univ. of Alberta, Edmonton, Canada, 1985.