CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

National Library of Canada
Bibliothèque nationale du Canada

Ottawa, Canada
K1A 0N4

TC –

ISE 0-315-23305-2

CANADIAN THESES ON MICROFICHE SERVICE – SERVICE DES THÈSES CANADIENNES SUR MICROFICHE

PERMISION TO MICROFILM – AUTORISATION DE MICROFILMER

• Please print or type – Écrire en lettres moulées ou dactylographier

AUTHOR – AUTEUR

Full Name of Author – Nom complet de l'auteur

GURMINDER SINGH

Date of Birth – Date de naissance
23 JAN 1958

Canadian Citizen – Citoyen canadien
[ ] Yes Oui [X] No Non

Country of Birth – Lieu de naissance
INDIA

Permanent Address – Residence fixe
CC/40A, G-8 RAJOURI GARDEN
NEW DELHI
INDIA - 110 064

THESIS – THÈSE

Title of Thesis – Titre de la thèse

'Presentation Component for the University of Alberta' CIMS

Degree for which thesis was presented
Grade pour lequel cette thèse fut presentee
M.Sc.

Year this degree conferred
Année d'obtention de ce grade
1985

University – Université
of Alberta

Name of Supervisor – Nom du directeur de thèse
Dr. Mark Green

AUTHORIZATION – AUTORISATION

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to **microfilm** this thesis and to **lend** or **sell copies of the film.**

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission

L'autorisation est par la présente accordée à la BIBLIOTHEQUE NATIONAL DU CANADA de **microfilmer** cette thèse et de **prêter** ou de **vendre des exemplaires du film.**

L'auteur se reserve les autres droits de publication ni la these ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur

ATTACH FORM TO THESIS – VEUILLEZ JOINDRE CE FORMULAIRE À LA THÈSE

Signature

Date
16 Sept 1985

NL-91 (r. 84/03)

Canada

The University of Alberta

PRESENTATION COMPONENT FOR THE UNIVERSITY OF ALBERTA UIMS

by

Gurminder Singh

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Fall, 1985

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Gurminder Singh

TITLE OF THESIS: Presentation Component for the University of Alberta UIMS

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: 1985

(Signed) .....

Permanent Address:
CC/40A, G-8 Rajouri Garden
New Delhi - 110 064
India

Dated 4 Sept. 1985

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Presentation Component for the University of Alberta UIMS** submitted by **Gurminder Singh** in partial fulfillment of the requirements for the degree of **Master** of Science.

Supervisor

Date: Sept. 4, 1990

ABSTRACT

This thesis is concerned with the design and implementation of the presentation component of the University of Alberta User Interface Management System. The Seeheim Model of User Interface Management Systems [Green1984a] is considered as the basis of the work.

It is shown that all device dependencies can be limited to the presentation component of the user interface. This increases the portability of the system. In the design proposed in this thesis user interfaces can easily be adapted to individual users. The selection of interaction techniques and display formats can also be easily changed to the actual user's liking.

It is observed that the existence of a separate presentation component encourages the use of standard libraries of interaction techniques and display procedures. This speeds up the process of generating user interfaces to a great extent and reduces the cost of programming considerably. A catalogue of interaction techniques is proposed. This catalogue defines the contents of the library of interaction techniques. The proposed classification scheme is based on the interaction task performed by the interaction technique.

Acknowledgements

I am indebted to my thesis supervisor, Dr. Mark Green, for introducing me to the problem, guiding me at each stage of the research, and offering valuable suggestions for the dissertation.

I would like to thank the members of my examination committee, Dr. Clifford Addison, Dr. Renee Elio, and Dr. Alinda Freedman for their valuable comments.

I am greatly indebted to my parents, to whom this thesis is dedicated, for instilling in me their belief in the value of education. Finally, I am grateful to all those who patiently waited for me to finish this work.

Table of Contents

List of Figures

Chapter 1

Introduction

Until recently most of software system designers have concentrated mainly on such characteristics of software quality as "portability", "reliability", and "testability" [Boehm1976]. Undoubtly these properties are desirable, but they are of primary interest to software engineers and managers. The goal of making the systems "user-friendly" remains secondary. 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 [Cheriton1976] 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 [Edmonds1981]. 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 [Buxton1983]. 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 thesis is a step in the direction of providing 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. Edmonds [Edmonds1982] defines the user interface as the part of the system that represents the user's model of the system. Krammer [Krammer1980] called the user interface the

"attention processor", whilst referring to the remaining system as a set of virtual "background processors".

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.

Despite the current interest in user interfaces, the design of good interfaces remains to a great extent an art, with much argument over guidelines and principles of interface design [Blesser1982]. Pertinent information is scattered throughout the literature of psychology, graphic design, linguistics, hardware design, and under the general umbrella of computer science. Until the concept of a user interface designer evolved the system designer was bogged down by the details of physical interaction handling, screen management as well as the design of the actual application. The end product was an application system with a patched interface.

The basic structure of a user interface does not change radically over a wide range of applications [Green1984b]. 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. What is A UIMS?

A UIMS is a collection of software tools supporting the design, specification, implementation, and evaluation of user interfaces [Seattle1983]. A UIMS serves to separate the design and specification of the interaction between the user and the application from the design and specification of the application functions that are to be invoked. 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, and freed from detailed concern with managing the flow of user actions and responses (Figure 1.1). UIMS has also been named a "Dialogue Management System" [Roach1982] or "Abstract Interaction Handler" [Feldman1982].

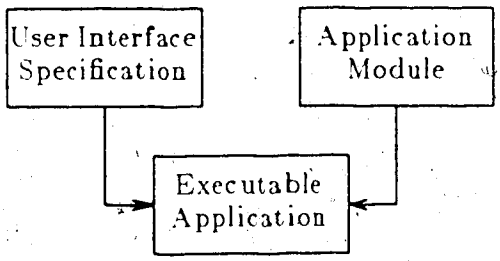

Figure 1.1 Generating an Application

Over the past few years many models of UIMSs have been proposed and implemented. The model considered in this thesis is the one proposed by a group headed by Mark Green at Seeheim Workshop on User Interface Management Systems [Green1984a]. In this model a user interface is divided into the following three components:

[1] Presentation Component: This is the device or physical level of the user interface. It 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. The device dependencies are limited to this component of the user interface; all the other components are device independent. The presentation component produces and consumes units of information called tokens.

[2] Dialogue Control Component: This is the main component of the user interface. It checks the sequence of input tokens produced by the presentation component for grammatical correctness and determines the operations the user wants to perform. This component contains the control logic of the user interface.

[3] Application Interface Model: This model represents the functionality of the application. It contains the description of the major data structures maintained by the application, and the application routines that can be invoked by the user interface. The application interface model serves two main purposes; first, it defines the interface between the user interface and the rest of the program; second, it contains enough of the semantics of the application to allow the UIMS to detect and possibly correct some of the semantic errors made by the user, before they are passed to the application.

A detailed description of various UIMS models appears in the following chapters.

1.3. The Presented Work

The research reported here is focussed on the issues involved in automatically generating the presentation component of the user interface. 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 thesis are based on the study of desirable features of user interfaces and the experience gained through the implementation of the system to generate the presentation component of user interfaces automatically.

1.4. Organization of the Thesis

The second chapter presents a survey of the research relevant to an understanding of user interfaces and UIMSs. It also presents the model of user interfaces considered as the basis for this research. The design of the system to generate the presentation component of the user interfaces automatically is described in chapter 3. Chapter 4 discusses the need for a catalogue of interaction techniques. The implementation details of the system to accept the specifications and generate the presentation component of a user interface are discussed in chapter 5. Chapter 6 contains a discussion of results and provides suggestions for future work.

Chapter 2

Background and Survey

This chapter describes the desirable properties of a user interface and presents a survey of the research relevant to the understanding of user interfaces and UIMSs. The model used as the basis of the U of A UIMS is also presented. Finally, a summary of objectives and efforts in implementing the U of A UIMS is presented.

2.1. Desirable Properties of A User Interface

Starting with the basic definition of the user interface given in Chapter 1, the next step is to define the desirable properties of a user interface. With our extremely limited knowledge of human behavior it is impossible to find a complete set of desirable properties of the user interfaces. Foley and Wallace have summarized some of the important features of user interfaces in [Foley1974]. Their suggestion is to design user interfaces so as to avoid boredom, panic, frustration, confusion, and discomfort to the user. Their paper also suggests some guidelines as to how to achieve this goal. Most of the guidelines are based on their personal experience and are not supported by any empirical evidence. Although, the paper lists some of the important properties of user interfaces it cannot be considered complete. The issue of effective use of color in user interface design, for example, receives little attention in [Foley1974]. As noted by Gerald Murch in [Murch1984], it is impossible to develop a complete set of guidelines for the effective use of color in all applications. Murch's paper, however, establishes some broad principles based on the mechanism of human color perception.

In designing interactive systems the effort should be to make the user-computer communication natural to the user. This naturalness in dialogue increases user productivity. One of the main guiding principles in skillful design of user interfaces relates to the communication language. The language of the dialogue must be the language of the user and context of the dialogue must be natural to him. It is also

essential for the language to be efficient and complete. An efficient language allows an effective and concise expression of ideas whereas a complete language permits expression of any idea relevant to the dialogue. The user interface designer must understand completely the user's task to be accomplished if a complete and efficient language is to be achieved.

The other guiding principle in the user interface design is to avoid psychological blocks like boredom, panic, frustration, confusion and discomfort to the user. Boredom and panic are related to the responses from the system. The user may get bored when the system takes a long time to respond to simple commands. Panic results because of unexpected delays wherein the system does not respond over a prolonged period of time. The remedy, when the real delay cannot be reduced, is to introduce trivial system responses providing assurance or acknowledging completion of intermediate steps.

Frustration results from the user's inability to convey intentions to the system easily. It is further compounded by inflexible and unforgiving systems by not providing functions like undo and error recovery. To avoid frustration to the user the system should provide recovery from incorrect actions, undo processing, and several ways of performing a task.

Confusion is a consequence of the system expecting or presenting unnecessary details. To avoid confusion to the user the system should organize information in a methodical manner and reduce unnecessary detail. Confusion can also be reduced by prompting and telling the user the tasks that can be performed. The user's attention should be directed to specific areas of interest on the screen by using different colors and/or blinking them.

Discomfort to the user may result because of inappropriate screen formats. For example, a badly designed menu may require excessive cursor movement from the user.

The system designers along with minimizing the cursor movement should take care of the left handed users.

In summary, user interfaces should be designed to minimize the time taken by the three types of basic human processes: perception, cognition, and motor activity. The perceptual process is the process whereby the user recognizes the information presented to him by the system. The cognitive process deals with how information is acquired, organized, and retrieved. The motor process comes into play when the user, having received, recognized, and decided how to respond, performs a response in physical actions.

2.2. 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 the 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.

A UIMS consists of software tools which help to generate, maintain, debug, and evaluate user interfaces automatically. The objective is to free the applications programmer from low-level details enabling him/her to concentrate on higher application-specific aspects of user interfaces.

2.3. Survey

During the past few years many models of UIMSs have been proposed and implemented. In this section a survey of some of the important existing UIMSs is presented.

2.3.1. TIGER

The UIMS described by David J. Kasik in [Kasik1982] is a part of The Interactive Graphics Engineering Resource (TIGER) which provides a framework for the development of engineering application at the Boeing Company. This system has its own programming language to define interactive dialogue sequences distinct from the application and a run-time module that specifically handles an end user's physical interactions.

In TIGER's view of an application, the user dialogue is separated from the algorithmic portions of each function. The user begins the execution of a process. The process can insure that it has necessary and sufficient information for its execution. An interactive function consists of three phases, namely, the input collection phase, the function execution phase, and the output modification phase. The output modification phase is required if the function determines that the given input results in multiple or ambiguous solutions or is in error. User dialogue is required only during input collection and output modification phases. The ultimate interpretation of the data that results from a function is the responsibility of the user. There are two major points to be noted about the TIGER UIMS. First, all interaction handling is accom-

plished through the user interface. Second, the user becomes the controller of the application. Figure 2.1 shows an overview of TIGER philosophy.

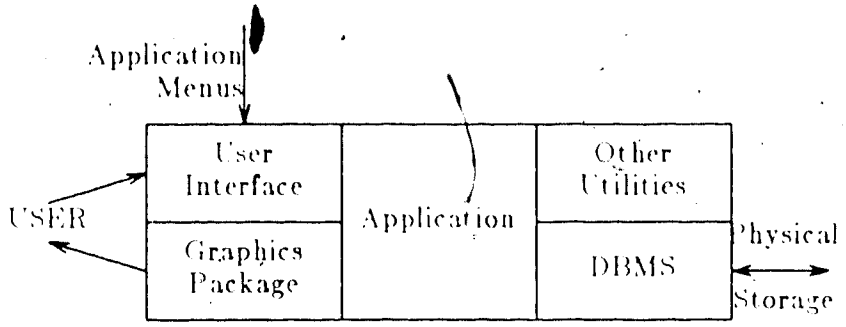

Figure 2.1 TIGER Application Architecture

To facilitate the description of the dialogue sequences and the input interrupt handling the TIGER UIMS provides two components. The first is a special purpose programming language called TICCL (TIGER Interactive Command and Control Language), which allows an application programmer to define the dialogue sequences. The second is the run-time interpreter which receives user inputs and passes parameters to the application procedures. A brief description of the TICCL and the run-time interpreter is presented in the following sections.

2.3.1.1. TICCL Description

This special purpose programming language provides the programmer with a tool to define and organize interactive dialogue sequences. A preprocessor compiles the dialogue sequences written in TICCL into a formatted menu file that is read by the run-time interpreter.

TICCL is a block structured language. For command organization the language provides a simple hierarchy of constructs. It also provides techniques to selectively enable input devices for parameter specification. The application can request the following data types at run-time:

- Entity identification (picking)
- Alphanumeric data
- Simple list of alternatives
- One value (for example, dial output)
- Two values (for example, tablet position information)
- Three values (for example, sonic pen position information)

2.3.1.2. Run-time Interpreter

The run-time interpreter portion of the UIMS assumes both input and output responsibilities for all interactive dialogue. It uses the results of the compiled TICCL language as input. The interpreter displays the command sequences to the user. The other graphical and geometrical information like position and style of display are all contained within the interpreter. All the information is presented to the user in the form of menus.

The run-time interpreter collects all the interrupts and processes them according to the constraints imposed by the static TICCL structure and dynamic inputs provided by the application at run-time. Interrupts that define input parameters to a function are queued by the interpreter without application intervention until a specific command is requested or a pseudo-command condition, such as, a carriage return is received. The interpreter passes specific information to the application via parameter lists that are specifically tailored to a particular input.

The UIMS provides the following additional capabilities:

- Command Set Display: TICCL allows the programmer to identify functions and sets of parameters as new levels in the language hierarchy. The run-time interpreter applies heuristics to examine the parameters and determine how far it can look ahead on a default path. All information on each level is displayed to the

user with the defaults clearly identified. The user is allowed to select any command below his current level, and the interpreter automatically executes the proper procedures to skip unselected levels.

- Implied Reject Processing: The user is allowed to select a command above the current level. During the normal interaction processing, this implies that the user is rejecting any already executed commands and wishes to follow a path that may or may not be displayed.
- Recursive Command Re-entry: The recursion is invoked by picking a command at a higher level than the one that the user is currently on. If the command is defined as re-enterable, a recursive call to the chosen function is generated.
- Situation Dependent Commands: Depending upon the specific context the user is presented with a set of legal commands. This helps in preventing errors.

The main problem with TIGER is that to be able to use the system the programmer must learn a new programming language (TICCL). The system also requires the programmer to enter geometrical information in textual form. The programmer's freedom in designing screen formats is limited as the form, positioning, and style of displays are all contained within the run-time module. TIGER's strength lies in its capability to handle complex interactive dialogue sequences. Its capability to handle implied reject processing and recursive command re-entry adds to its power as a practical tool.

2.3.2. U of T UIMS

This system was developed in 1983 by W. Buxton, M. R. Lamb, D. Sherman, and K. C. Smith at the University of Toronto [Buxton1983]. The UIMS consists of two main modules. The first is a preprocessor which enables the program designer, using interactive graphics techniques, to design and specify the graphical layout and functionality of menu-based user interfaces. It is with this module that the applications programmer establishes the relationships between user actions and the application specific routines.

The second main module is the run-time support package. This package handles event and hit detection, procedure invocation, and updating the display according to the schema specified using the preprocessor. The code executed by the run-time module can be automatically generated from the data specified to the preprocessor.

2.3.2.1. MENULAY and MAKEMENU

The preprocessor which serves as the front end of the UIMS is known as MENULAY. The package is designed to enable the user interface designer to specify the graphical and functional relationships within and among the displays making up a menu-based system. MENULAY enables the designer to define user interfaces which are made up of networks of menus. It has two levels of use: novice and expert. In novice mode, only the basic commands are accessible. All input is done through one button on the cursor puck, and instructions are given at various points in the program. In expert mode, the other three buttons on the puck can be used to perform special functions, and fewer instructional diagnostics are displayed.

The specifications made by using MENULAY are converted into the C programming language and compiled through the use of a companion program MAKEMENU. The resulting code can be linked with the application-specific routines. The entire sequence of creating an interactive dialogue is shown in Figure 2.2. The programs

generated by MAKEMENU contain unresolved external references ("hooks") where the applications designer specified names of application-specific functions. By writing functions with the required names and referencing the appropriate file names when MAKEMENU is called, the applications programmer can add any amount of application-specific programming to the layout and sequence information specified by the designer.

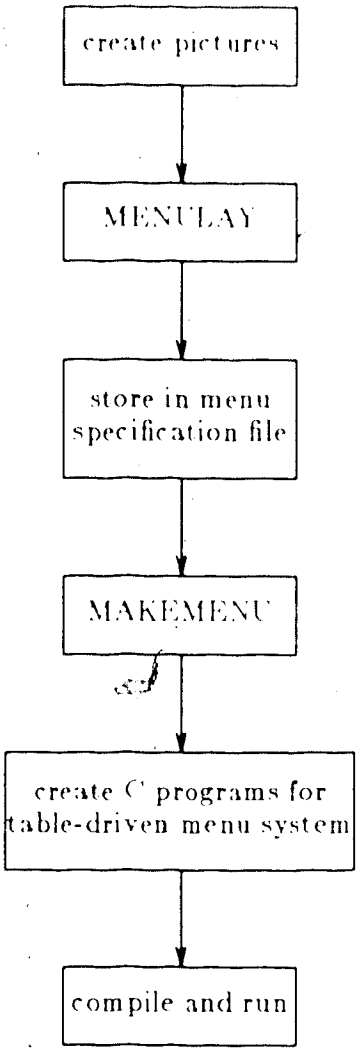

Figure 2.2 A Sequence for Constructing an Interactive Dialogue .

The U of T UIMS provides facilities for entering geometrical information graphically and integrating the design specifications with hand-coded application routines.

The structure of the UIMS facilitates the generation of code in different programming languages. To output code in a different language would involve rewritting MAKEMENU and providing run-time support in a compatible format. The UIMS does not allow the designer to create more than one window. This restricts the structured design of user interfaces.

2.3.3. SYNGRAPH

The SYNGRAPH (SYNtax directed GRAPHics) user interface generator was developed as part of the AHI (Automated Human Interfaces) project at Arizona State University by Dan R. Olsen Jr. and Elizabeth P. Dempsey in 1983 [Olsen Jr.1983a]. It generates interactive Pascal programs from a description of the input language's grammar. The system concentrates on the use of formal languages for implementation rather than specification. A SYNGRAPH interface specification is decomposed into conceptual, semantic, syntactic, and lexical levels. The conceptual level guides implementation rather than being part of it and is therefore only indirectly part of such specification. The interaction semantics of SYNGRAPH are viewed as a set of procedures and data types written in the implementation language. The syntactic level is described using an extended version of BNF (Backus Naur Form) which includes tokens, semantic actions, and information about how the user interface is organized. The lexical level binds the logical token names to actual input devices and techniques. A description of lexical, syntactic, and semantic levels appears in the following sections.

2.3.3.1. Lexical Specification

In the lexical specification, various input devices and techniques are bound to logical token names. The token names are then used in the syntactic specification. The mapping of this token set to a set of interactive resources is handled in the syntactic specification.

There are seven primitive input devices supported by SYNGRAPH. They are: menu items, the locator, valuators, function buttons, keys, characters, and picks. Menu items are defined simply by giving a name without any further specification. All such tokens are automatically organized into menus as defined by the syntax specification. The difference between key and character tokens is that a key token refers to a specific character and returns no value, whereas a character token refers to a character and returns the character as its value. The key token's primary purpose is to construct keywords and delimiters to control the parse. The purpose of a character token is to input textual material. Each primitive device has the following predefined properties:

- The type of value it returns.
- Whether it is an event or sampled input.
- The display action required when it is enabled.
- The prompt required when it is a legal input in the current parse state.
- The acknowledgment required when it has been selected.
- What effect it has on the choice of parse transitions.

2.3.3.2. Syntactic Specification

The syntactic specification specifies the sequence of tokens constituting valid input streams along with the organization and arrangement of their prompts and echos. The syntax of the dialogue is expressed in terms of a modified BNF which uses the logical token names defined in the lexical specification as well as non-terminals declared in the grammar. The grammar used by SYNGRAPH allows for optional phrases, repeating phrases zero or more times (curly braces {}) as well as alternation (vertical bar |). These extensions beyond BNF allow non-terminals to be used to organize the dialogue.

In complex interactions the logical resources required are much larger in number than those actually available. It is therefore necessary to decompose the user interface into smaller sub-dialogues. The resources are then allocated within each sub-dialogue. The appropriate choice of such an organization must be application-dependent so that functionally similar activities can be grouped together. This organization is expressed by designating certain non-terminals as defining new levels of interaction. A level of interaction extends from the non-terminal which is specified as its head through all non-terminals invoked by it either directly or indirectly which are not new levels themselves. When a new level of interaction is entered all of the tokens which are used within that level become enabled. In the case of menu items, simulated buttons and simulated valuators, this means that they are allocated screen space and displayed. A level therefore defines a specific configuration of virtual input devices.

Most of the prompting of the user is performed automatically by SYNGRAPH. This is done by examining the set of input tokens that are acceptable for a given state. Feedback is also performed automatically when a particular input has been selected. In addition to the automatic prompting, it is possible to add additional information to the grammar. Help items can be added as special transitions which do not effect the

parse.

2.3.3.3. Semantic Specification

The SYNGRAPH system is written in Pascal and the interaction semantics are handled using Pascal statements. Each non-terminal is generated as a recursive Pascal procedure whose control structure is the syntax tables produced from the input grammar. Each non-terminal can have a parameter string and a header string associated with it. The parameter string must be a Pascal parameter specification and the header string a set of Pascal declarations. Within the production for the non-terminal, semantic actions are expressed by Pascal statements. When the parse reaches the state where such an action appears, it is executed.

To be able to use SYNGRAPH the designer has to learn a new programming language. The system also requires the designer to represent geometrical information in textual form. This approach is similar to the one followed in TIGER. The allocation and organization of menus and virtual devices is outside of the user's control. For a more natural interaction these should be user controllable. The important features of SYNGRAPH include automatic prompting, highlighting, and feedback.

2.4. The Seeheim Model of User Interfaces

The Seeheim model of user interfaces [Green1984a] is the one considered in this thesis. In this model a user interface is divided into three components as shown in Figure 2.3. The presentation component is responsible for the external presentation of the user interface. This component generates the output and reads the physical input devices. The dialogue control component defines the structure of the dialogue between the user and the application program. The application interface model defines the interface between the user interface and the rest of the program. It is the representation of the application from the view point of the user interface.

2.4.1. Presentation Component

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.

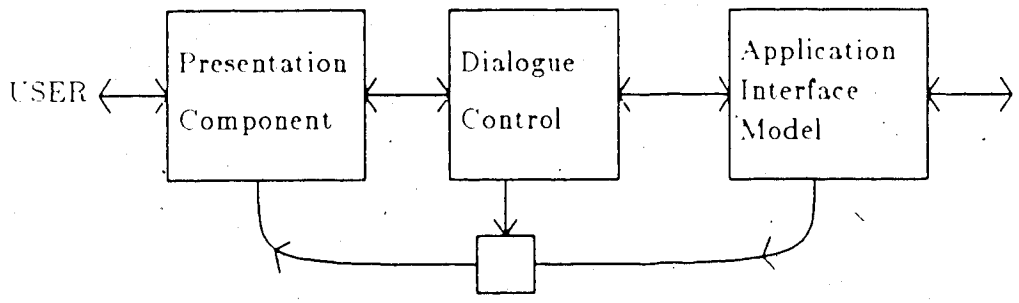

Figure 2.3 The Seeheim Model of a User Interface

The main purpose of the presentation component is to translate between the external physical representation of the tokens, and their internal abstract representation. In most cases this translation is a simple one-to-one mapping. This mapping determines how the user's actions are converted into input tokens and how output tokens are converted into displays. The external-internal mapping can be viewed as a dictionary, with one entry for each of the external and internal data items. This entry indicates how the token is to be translated. The presentation component has no control over the contents of the dictionary, it cannot change the external-internal mapping.

There are a number of advantages of having a separate presentation component. Si[illegible] the device dependencies are limited to this component, the other component's

of the user interface can be moved to a different device without any modifications. This increases the portability of the user interface. The presentation component can be designed to support a range of devices and automatically adapt to the one in use. A separate presentation component provides a convenient means of adapting the user interface to individual users.

2.4.2. Dialogue Control Component

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 largely on this component of the user interface.

2.4.3. Application Interface Model

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 description of an application's data structure includes the type of information stored and its structure. The description of an application routine includes the

name of the routine and the number and types of its parameters. The routine descriptions might also include pre- and post-conditions. The user interface may communicate with the application in one of the three possible interaction modes. In the first interaction mode, the user initiated mode, the user calls the routines in the application. In the second type of interaction, called the system initiated mode, the application calls routines in the user interface. The third interaction mode, mixed initiative, is based on two communicating processes, one for the user interface and one for the application. The user interface designer specifies the interaction mode and the UIMS establishes the procedures required to implement it.

2.5. The University of Alberta UIMS

The University of Alberta UIMS [Green1985] is based on the Seeheim model of user interfaces discussed in section 2.4 of this chapter. There are a number of reasons for developing this UIMS. First, we wanted to evaluate the feasibility of the Seeheim model as the basis for the UIMS. Since this model had not been used as the basis for any other UIMS, there was no evidence of its usability. Second, we wanted a test bed for testing new ideas in user interface design and implementation. Third, we wanted a practical tool that could be used in other research projects within our department.

The design and implementation details of the presentation component of the U of A UIMS are described in this thesis. For specifying the dialogue between the user and computer three main notations are considered. These notations are recursive transition networks, BNF grammars, and events. The system to accept the dialogue specified using recursive transition networks and convert it into an intermediate code, called EBIF (Event Based Internal Form) is discussed in [Lau1985]. Details about the event language and its implementation can be found in [Chia1985]. At the present time the implementation of the system for grammar based notation has not been started. The support for the application interface model is under developement.

Chapter 3

The Design

This chapter describes the structure of the presentation component of the University of Alberta UIMS and explains the related concepts.

3.1. 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 3.1. The input tokens are the tokens which flow from the user towards the application and output tokens flow from the application towards the user. The user interactions (called events, as described in chapter 5) flow from the input devices to the interaction techniques. An interaction technique manipulates the raw data generated by the interactions to produce more meaningful information. The control module then restructures this information into an input token and sends this token to the dialogue control component of the UIMS. In the case of output tokens, the control module assigns the output token to a display procedure and the window where its image is ultimately generated.

In the following sections a description of the important concepts related to the presentation component of the University of Alberta UIMS is presented.

3.2. 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

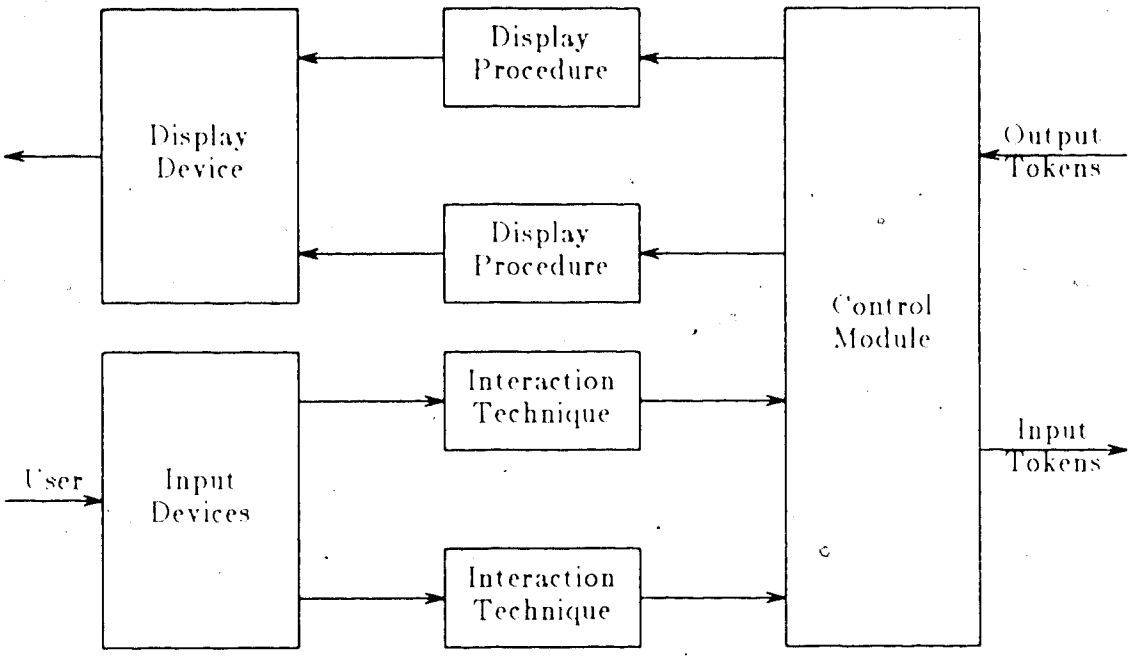

Figure 3.1 The Structure of the Presentation Component

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 input token by the presentation component of the UIMS. The interpretation of the token value depends upon the token number. Often, the token value field points to a structure containing the following information.

- X coordinate value
- Y coordinate value
- Value

3.3. Output Tokens

The output tokens are used for generating images. These tokens are generated by the dialogue control component as well as the application. They 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 makes sure 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 output token by the presentation component of the UIMS. Using the token number as the key the control module finds out 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.

3.3.1. Token Definition File

This file contains the definitions of the structures the value field of an input or output token could point to. The definitions in this file are shared by the presentation and dialogue control components of the user interface and the application routines. This helps in maintaining the consistency between the information passed and its interpretation by the presentation component. It also makes the organization more structured in the sense that all the structure definitions are located in one file and the changes are consistently conveyed to every component using the file. This file is updated only when there is an update in the library of display procedures. The library of display procedures is explained in the following sections.

3.4. 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 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.

3.5. Interaction Techniques

Most interaction tasks are carried out or implemented 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 [Foley1981].

Each interaction technique has a specific purpose, such as to specify a command, designate a position, or select a displayed object, and each is implemented with some device, such as a tablet, joystick, keyboard, or light pen. Typical techniques include specifying a value using a potentiometer, designating a displayed object with a pointing device, or specifying a position with a tablet. Each interaction task can be implemented by many interaction techniques. The designers of the system should select interaction techniques that best match both the user's characteristics and the specific requirements of the task being performed.

3.5.1. Library of Interaction Techniques

There is a multitude 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. This way one can keep building the library and help reduce the cost and time of producing good user interfaces.

3.6. 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. Each display procedure has a specific purpose and is used to generate a specific image. The display procedures expect information to be passed to them in a predefined format. The display procedures also assume the availability of the particular type of display device on which the procedures can generate images. Examples of display procedures are Angle Display, Vertical Bar Display, and Text Windows.

3.6.1. Library of Display Procedures

The idea of having a library of display procedures is similar to that of the library of interaction techniques. By adding the 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.

Chapter 4

A Catalogue of Interaction Techniques

This chapter explains the need for having a catalogue of interaction techniques and discusses its advantages. The general format for a catalogue entry is explained. The specification language GUSL (Graphical User interface Specification Language) is used for formally specifying the interaction techniques. A brief introduction to GUSL is also presented.

4.1. Introduction

The designer of an interactive graphics system is faced with a difficult task of deciding on the interaction techniques that best match both the user's characteristics as well as the system requirements. Quite often when faced with such a problem the designer develops his/her own interaction techniques or tries to adopt one from a small set of familiar interaction techniques. The designer is normally unaware of the interaction techniques successfully used by others. In order to make the designer's choice wider a library of interaction techniques has been proposed in chapter 3. To be able to decide about which interaction technique to use, the designer must have the complete description of each of the interaction techniques in the library. The catalogue of interaction techniques is intended to be such a source of information. The catalogue not only provides information about the computational/graphical aspects of the interaction techniques but also human factors considerations. It can be considered as an important source of guidance when selecting an interaction technique. Similar ideas about producing a catalogue of interaction technique can also be found in [Green1983a], [Foley1981] and [Foley1984].

To be useful the catalogue should be organized in such a way that the entries for interaction techniques performing similar interaction tasks appear together. The catalogue should provide a concise functional description of each interaction

technique. It should also include the initializations, data required, output produced, interaction task performed, and the hardware prerequisites to use the interaction techniques. A precise formal specification of each of the interaction techniques should also appear in the catalogue entries. Finally, the catalogue should also describe the ergonomic quality of the interaction techniques.

4.2. The Catalogue of Interaction Techniques

The catalogue of interaction techniques consists of a number of entries each of which corresponds to one interaction technique. The general format of a catalogue entry is shown in Figure 4.1. The first field of the entry is the name of the interaction technique. The name normally consists of several words describing the function of the interaction technique. The words comprising the name are used in locating interaction techniques by keyword lookup. This lookup mechanism is explained in the following sections.

Name:
Classification:
Initialization Parameters:
I/O Parameters:
I/O Devices
Description:
Specification:
Ergonomics:

Figure 4.1 General Format of a Catalogue Entry

Each interaction technique is assigned a unique classification code. This code appears in the classification field of the catalogue entry. The classification code is a multi-level code consisting of a sequence of alphanumeric characters. The catalogue can be searched for interaction techniques belonging to a particular class. The classification scheme will be explained in detail in the following sections.

The next field of a catalogue entry describes the interaction technique's parameters. These parameters are used for initializing the instance of the interaction tech-

nique being used. Each parameter definition consists of a parameter name and a type. Since these parameters are used for initializing the interaction technique their values must be supplied before the interaction technique is used. The output produced by the interaction technique depends upon these parameters. The interaction technique by itself does not change these parameters.

The input/output parameter field of a catalogue entry describes the input parameters required every time a call to the interaction technique is made, and the output values produced by the interaction technique. Each parameter definition consists of a parameter name, its type, and whether the parameter is input or output.

The input/output devices field of a catalogue entry describes the hardware device requirements of the interaction technique. This field also lists alternate devices that can be used.

The description field of a catalogue entry contains an informal description of the interaction technique. This description covers the functionality of the interaction technique, how it interacts with the user, and how it treats its input/output parameters. It does not describe the implementation details. The description is detailed enough for the designer to determine if the interaction will fulfill the requirements.

The specification field consists of a formal specification of the interaction technique. Since the description of the interaction technique contained in the description field may be brief and fairly vague, a precise statement of how the interaction technique is implemented and what function it performs is required. This is described using a formal specification language called GUSL (Graphical User interface Specification Language). An introduction to GUSL is presented later in this chapter.

The ergonomics field contains the information about the human factors of the interaction technique. This data serves as a guide to the designer as to the appropriate use of the interaction technique. The information contained in this field should

provide measures of cognitive load, perceptual load, and motor load for the interaction technique. Unfortunately the absolute measures for these loads for most interaction techniques are not available [Foley1984]. The ergonomic quality of the interaction technique if possible, should be compared with the other interaction techniques performing the same task.

4.3. Classification of Interaction Techniques

The interaction techniques are classified according to the interaction tasks performed by them. The classification code is a multi-level code where each successive level in the code provides more specific information about the interaction technique. Under such a scheme the classification codes for functionally equivalent interaction techniques can be similar.

The classification scheme is based on four fundamental types of interaction tasks. These interaction tasks are: selection, position, quantify, and text input. In a selection task, the user makes a selection from a set of alternatives. The set might be a group of commands or a collection of displayed objects. In a positioning task, the user positions an entity at a particular position on the display. In a quantify task, the user specifies a value to quantify a measure. In case of a text input task, the user enters a text string. It is important to note that the text string entered under this task itself represents the information to be stored in the computer. This text string is not used for representing a command, a value, or a position. The text input is a new interaction task, not an intermediary step in one of the other interaction tasks. Each of these interaction tasks can be implemented by a number of interaction techniques. It may be noticed that the interaction tasks do not depend on the hardware or logical devices implementing them whereas the interaction techniques do. The first level of the classification code is the interaction task performed by the interaction technique.

The second level of the classification code provides information about the dimensionality of the interaction technique. For example, the interaction technique performing the positioning task in 2D has 2 as the value for the second level in its classification. This level contains an NA (Not Applicable) for interaction tasks for which dimensionality is irrelevant.

The third level of the classification code provides information about the type of feedback provided by the interaction technique. The feedback may be discrete or continuous. In the case of discrete feedback, the feedback is provided once on completion of the task. The feedback is provided all through the duration of the interaction in the case of continuous feedback. For example, a positioning task could be implemented with a continuous or a discrete feedback.

The last level of the classification code identifies the virtual device used for implementing the interaction task. An interaction task can be realized by one or more of just four distinct virtual devices, the pick, the button, the locator, and the valuator. For example, a selection task can be implemented by a pick or a button. We consider virtual devices, rather than the actual hardware devices as part of the classification code for the following two reasons. First, each hardware devices can be treated as a physical realization of one or several virtual devices [Foley1974]. Second, a change in hardware devices does not necessarily represent a change in the basic implementation or the user's perception of the interaction technique. For example, user's view of the interaction technique for menu selection does not change whether a mouse or a tablet is used in its implementation.

A pick is used to designate user defined objects like a menu item, a window, or a command. The lightpen is the prototype pick and can be conveniently used to point at displayed objects. A button is used for selecting system defined objects. Prototypes for this device are the programmed function keyboard and the alphanumeric

keyboard. A locator is used to indicate a position and/or orientation in the user's conceptual drawing space. It is typified by the tablet, joystick, and mouse. A valuator is used to input a single value in the real number space. A model for this device is a potentiometer.

As an example, the classification code for an interaction technique implementing a positioning task in 2D and providing continuous feedback would be Position.2.Cont.Locator.

4.4. Introduction to GUSL

The specification language GUSL [Green1982] is based on the state machine approach of Parnas [Parnas1972]. In the state machine approach to specification a program is divided into a number of state machines. Each state machine has a local state and a number of functions that can be used to access or change its state. The state machines in GUSL are called modules.

A GUSL module has four components. The first component is the declarations of the module's parameters. The second component contains the variable declarations for the module. The third component of a module is the definitions component. The fourth component of a module contains the definitions of its functions. In the first component the module parameters are used to define the initial state of the module. This component starts with the keyword PARAMETERS followed by the parameter declarations. A parameter declaration consists of a parameter name, a colon, and the type of the parameter. The type can be any of the primitive GUSL types or the name of another module. The primitive GUSL types are integer, real, string, set, boolean, point, and extent. The point is used for representing two dimensional points and extent is used for representing area in a two dimensional plane. An example parameter component is:

```
PARAMETER
    a, b : real;
    obj : geo_obj;
```

The type of the parameter obj is the name of another module called geo_obj. All the functions defined in the geo_obj module can be applied to this parameter.

The second component contains the variable declarations for the module. It starts with the keyword DECLARATIONS followed by the individual variable declarations. A variable declaration has essentially the same format as a parameter declaration. The only difference occurs when a module name is used as a type. In the declarations component values must be supplied for that module's parameters. Each time a module name is used as a type in the declarations component a new instance of that module is created with its own local state. An example declarations component is:

```
DECLARATIONS
    start, end : point;
    symbol : geo_obj( "circle" );
```

The definitions component contains the definitions of the syntax macros used in the module. Syntax macros are used to shorten functions definitions and make them more readable. This component starts with the keyword DEFINITIONS followed by the syntax macros declarations. A syntax macro declaration consists of the type of the result, the macro's name and parameters, and the expression that defines the macro. Some example macro declarations are:

```
DEFINITIONS
    boolean x_range( p ) IS 0 < p.x < max_x;
    real m IS (end.y - start.y) / (end.x - start.x);
```

The first macro, x_range, is used to determine if the x component of its parameters is within a certain range. The second macro returns the slope of the line between the two points start and end.

The functions component is made up of the function declarations for the module. This component starts with the keyword FUNCTIONS followed by the individual function definitions. There are two kinds of functions in GUSL, namely, the V function and O function. Each kind of function has its own definition format.

The V functions are used to represent and access the state of the module. The definition of a V function is divided into three parts; function header, pre-conditions, and function body. The function header consists of the keyword VFUN, the name of the function, its parameter declarations, and the type of the result. The parameter declarations have the same format as module parameter declarations. The pre-conditions part of the function definition starts with the keyword PRE followed by a list of logical expressions. All the expressions in the pre-condition part must be true before the function is invoked. The format of expressions in GUSL is discussed in detail later in this section.

There are two types of V function bodies. In a primitive V function, the function body gives the initial value of the function. This value can be changed by the O functions in the module. Primitive V functions are used for representing the state of the module. The function body of a primitive V function can be prefaced with the keyword HIDDEN in which case the function can only be accessed within the current module. The derived V functions provide the interface between the internal state of the module and the other modules in the specification. The body of this type of function consists of the keyword DERIVED followed by a sequence of expressions. The last expression in the sequence must equate the function name to a value.

The O functions are used to change the state of the module. They do so by

changing the value of the primitive V functions. The definition of an O function is divided into three parts: function header, pre-conditions, and post-conditions. The function header for an O function is similar to that of a V function, except that there is no result type. The pre-condition part of an O function is same as the corresponding part of a V function. All the expressions in the pre-conditions part must be true before the function is invoked. The post condition part of an O function consists of the keyword POST followed by a sequence of expressions. When the O function is invoked these expressions are considered one at a time starting with the first one. For each expression the internal state of the module is modified to make the expression true. The O function does not specify how these modifications are made.

The expressions in GUSL are logical statements that evaluate to the values true or false. These statements consist of the variables, parameters, functions defined in the module, boolean operators, functions defined in the other modules, and special forms. If the type of a variable or parameter is the name of another module then any of that module's functions can be used in expressions involving that variable. There are two formats for this type of function reference:

variable_name.function_name(parameters)

operand function_name variable_name

In either case function_name is the name of a function defined in the module used as variable_name's type. The second format can only be used with functions that have one argument.

There are three special forms that can be used in GUSL expressions. The LET special form has the following format:

LET variable_name : type_name | expression;

The special forms create a new value of type type_name for the variable variable_name. This new value must satisfy the expression in the body of the special form.

The format for the FORALL special form is:

```
FORALL variable_name : type_name | p( variable_name )
        {
        expressions
        };
```

In this special form variable_name is a variable local to the special form and of type_name. In order for this special form to be true all values of the variable satisfying the predicate p must also satisfy the expressions in the body of the special form.

The EXISTS special form has the following format:

```
EXISTS variable_name : type_name | p( variable_name )
        {
        expressions
        };
```

In this special form the variable variable_name is local to the special form and of type type_name. This special form has the value true if at least one of the variable values that satisfy the predicate p also satisfies the expressions in the body of the special form.

4.5. An Example Catalogue Entry

The graphical potentiometer is being taken as an example to show the contents of a typical catalogue entry. The catalogue entry for this interaction technique is shown in Figure 4.2.

Name:

Graphical Potentiometer

Classification:

Quantify.NA.Discrete.Locator

Initialization Parameters:

llx, lly, urx, ury : real

min_value, max_value : real

orientation : integer

I/O Parameters:

x, y : real, input

value : real, output

I/O Devices:

Any input device that can be used as a 2D tablet locator can be used here. The best choices are either a tablet or a mouse.

Description:

This interaction technique is used for entering a real numeric value. The potentiometer can be set up in horizontal or vertical orientation. If the value of the parameter "orientation" is one the potentiometer is set up in horizontal position otherwise in vertical position. The user can interact with the potentiometer by

moving the graphical cursor in the area allocated for the potentiometer and generating an event. On receiving the event the potentiometer generates a real numeric value. The value of the potentiometer is constrained to be in the range min_value <= value <= max_value. The initial value of the potentiometer is min_value.

Specification:

```
MODULE g_pot;
  PARAMETERS
    llc, urc : point;
    min_value, max_value : real;
    orientation : integer;
  FUNCTIONS
    VFUN area -> extent;
      INITIALLY
        area = ?;
    END;

    VFUN inp( p : point ) -> boolean;
      DERIVED
        LET r : boolean | p in area;
        inp = r;
    END;

    VFUN ret( p : point ) -> real;
      DERIVED
        LET val : real |
         IF orientation = 1 THEN
           val = ((max_value-min_value)/(urc.x-llc.x))* (p.x - llc.x);
```

```
        ELSE
          val = ((max_value-min_value)/(urc.y-llc.y))* (p.y - llc.y);
        ENDIF;
      ret = min_value + val;
    END;

    OFUN init;
      POST
        IF orientation = 1 THEN
          line(llc.x, (llc.y+urc.y)/2,
              urc.x, (llc.y+urc.y)/2 );
        ELSE
          line((llc.x+urc.x)/2, llc.y,
              (llc.x+urc.x)/2, urc.y );
        ENDIF;
        LET e = extent( llc, urc );
        area = e;
    END;

END MODULE g_pot;
```

Ergonomics:

The experimental results for the measures of cognitive, perceptual, and motor processes are not available. The interaction technique is very easy to use and can be mastered by a user at the operators level after a few trials.

Figure 4.2 A Typical Catalogue Entry

4.6. Using the Catalogue

The catalogue consists of one entry per interaction technique. The catalogue is arranged according to the interaction tasks performed by the interaction techniques. This has the advantage that the designer while looking for guidance can find the information about similar interaction techniques together.

The catalogue can provide a very fast and effective source of information if stored on a computer. A set of programs similar to "apropos" and "man" running on UNIX† can be made to work in association with the catalogue. The catalogue can then be accessed based on the name of the interaction technique. The designer may also search the catalogue for entries belonging to a particular class. In this case the designer is provided with the names of all the interaction techniques belonging to the specified class. The search can be based on the information stored in the name and classification field of the catalogue entries.

† UNIX is a trademark of AT&T Bell Laboratories

Chapter 5

The Implementation

This chapter describes the implementation of the presentation component of the University of Alberta UIMS. A brief description of the graphics and data base packages used in the implementation is also presented.

5.1. Environment for 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.

5.2. 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 thesis. 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 "ipes" (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 "peg" (presentation component generation), consists of a number of procedures which provide run-time support and generate the presentation component of the user interfaces. This part is driven by the data base and the token tables. The "C" procedures produced by ipes are compiled and linked with the peg procedures. The entire sequence of creating a presentation component is shown in Figure 5.1.

In implementing the presentation component a graphics package called WINDLIB and a data base package called FDB are extensively used. A brief description of WINDLIB and FDB appears in the following sections.

5.3. Introduction to WINDLIB

WINDLIB [Green198?c] is a resource based graphics package. It is capable of driving multiple input/output devices simultaneously and dividing the screen of one display into several independent areas called windows. A window defines an abstract coordinate space and maintains a set of attributes such as window limits, drawing colour, and background colour etc. The window attributes define how and where the picture is displayed. Several windows with a common ancestor can overlap on the screen. The windows on each device have a priority ordering. The first window created has the lowest priority and the most recently created window has the highest priority. The program can select between overlapping and non-overlapping windows. In the case of overlapping windows the window priority is used to determine which

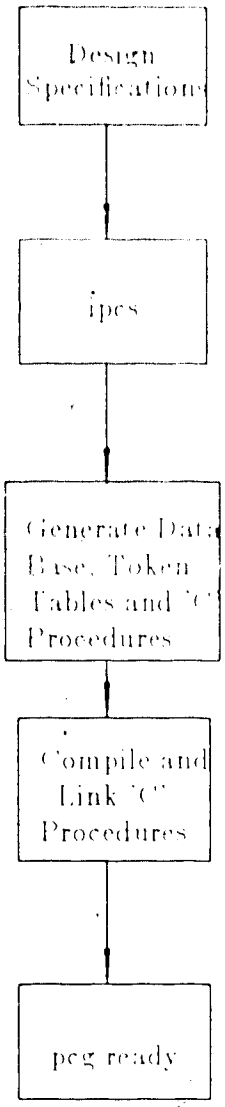

Figure 5.1 A Sequence for Constructing a Presentation Component

window is visible when several windows overlap.

The input system in WINDLIB is based on events and event handlers. An event can be generated by an input device or one of the routines in the program. Each event consists of an event type, a position (in 2D), and the name of the window it occurred in. The event is sent to the window with the highest priority that contains the event position. When a window receives an event, its event handler is invoked with the event as an argument. Thus each window is responsible for handling all the user actions that occur within it. Since each window behaves like an independent display

and handles all input directed at it, it is possible to partition the presentation component into a number of independent logical units which can interact amongst themselves if required.

In WINDLIB, anything which can be drawn in a window can also be stored in a contents structure. A contents structure is basically a hierarchical model where, in addition to the primitives provided by WINDLIB, programmer-defined primitives are also allowed. The contents structures can be used as a part of an output token to send information to the presentation component. The contents structure separates the generation of data, which is the responsibility of the application routines, from its display, which is the responsibility of the presentation component.

5.4. Introduction to FDB

FDB (Frame Data Base) [Green1983b] is a data base oriented towards scientific and engineering applications. The database is capable of handling a large number of entity types and representing complex structuring of information. The database can also represent entities with incomplete information.

The basic organizational unit in FDB is called a frame. A frame is made up of a collection of slots. Each frame in FDB has three standard slots called "owner", "type", and "isa". The owner slot stores the owner of the frame. The owner of a frame is the user who created the frame. The type slot indicates whether the frame is an ordinary frame or a meta frame. An ordinary frame is used for the storage of data. A meta frame is used to describe the structure of other frames in the database. The meta frames define the default slots and their values. The default values stored in the slots in ordinary frames can be changed. The value of the "isa" slot in a frame is the associated meta frame for that frame; that is, this slot is used to link ordinary frames to their meta frames.

Apart from the standard slots a frame also has other slots for storing data. Each slot is capable of storing one piece of data. A slot has four fields: "name", "visibility", "type", and "value". The name field of a slot contains the name used to reference that slot. The visibility field of a slot can have a value between 0 and 255. A visibility 0 can be seen by all users of the database. Each database user has a visibility level; slots with visibility greater than this level cannot be seen by the user. The type field contains the type of value stored in slot. In FDB the primitive values are integers, real numbers, strings, and frame. The frame data type is used to store the name of a frame. The value field contains the actual value in the case of integers and a pointer to the actual value in the case of other data types.

5.5. 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

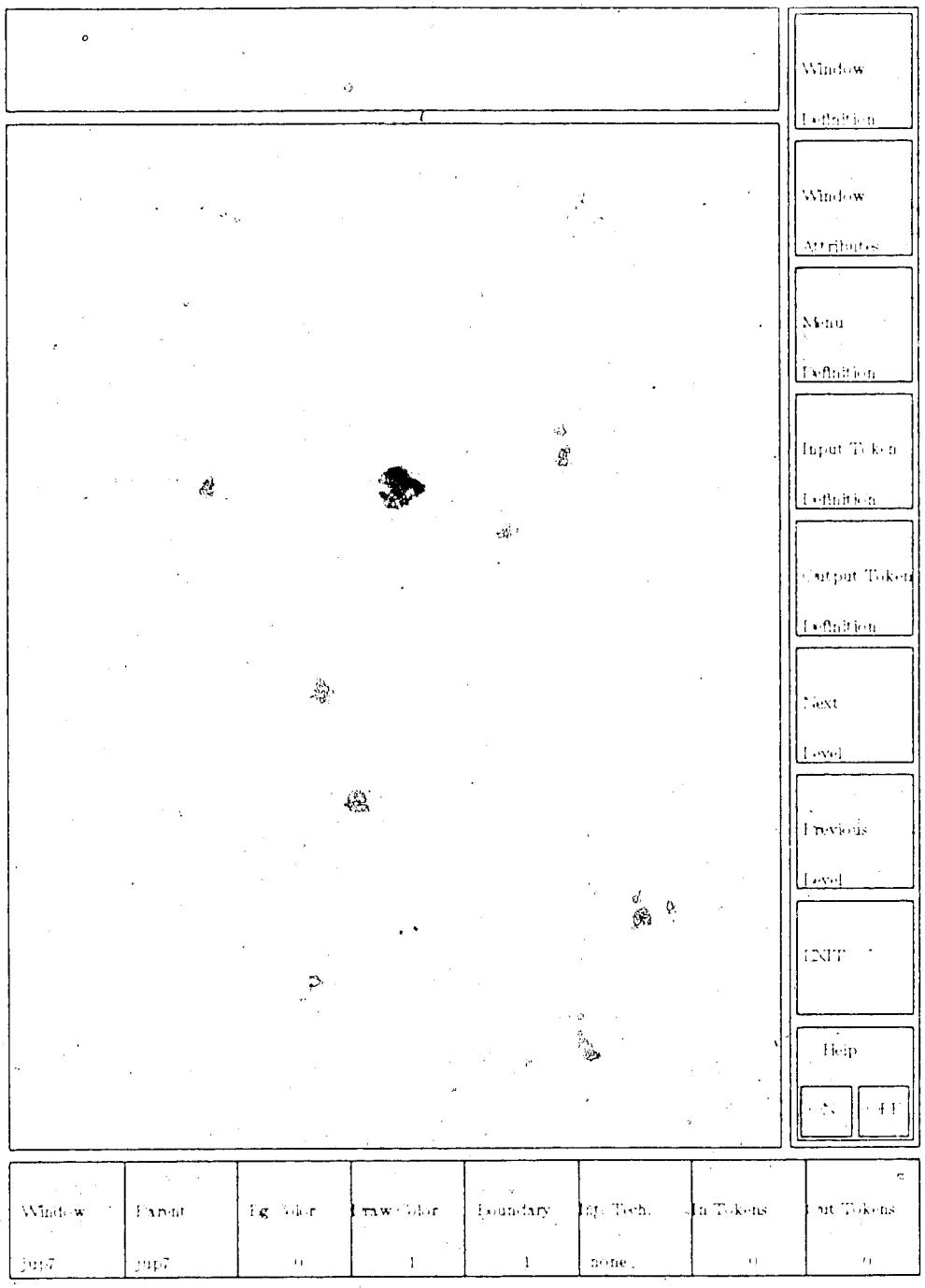

Figure 5.2 The ipcs Screen Layout

The interactive specification program "ipcs" is used to enter the design information. The ipcs screen is divided into four areas as shown in Figure 5.2. 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 ipes 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 ipes is presented.

5.5.1. Window Definition

Ipes starts off by displaying the layout shown in Figure 5.2. In the beginning 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 has to select the "Window Definition" command from the ipes 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 a level.

5.5.2. Window Attributes

The window attributes can be defined by selecting "Window Attributes" command from the ipes 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. Along with specifying the interaction technique the designer also specifies the input event generated by the interaction technique. The event numbers are specified in the call to the procedure implementing the interaction technique. The input tokens generated by these events

are specified using the "Input Token Definition" command. It is the responsibility of the interaction technique to generate only those events which are specified in its parameters. The run-time support module will generate an error message on receiving an event which does not appear in any input token definition.

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. Window name is used by the designer for his/her own reference. A meaningful window name helps to remember the important functions of the window.

Ipes 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. The display for such 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 a 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 pre-determined 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 designer-defined attribute. This facilitates the interactive design of user interfaces. The easy mechanism for accomodating 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.

5.5.3. Tree of Windows

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 designer can examine the contents at various levels of the tree by using the "Next Level" and "Previous Level" commands. The system allows the designer to modify the tree of windows as often as desired. The branches in the tree can be added or deleted by 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. The flexibility in designing the tree allows the designer to work more methodically and concentrate on one part of the user interface at a time.

5.5.4. 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", it can be changed to "pop-up" by typing "p" inside the box displaying the attribute. The placing of various 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" by typing "d" inside the box displaying the option. The default menu orientation is "vertical", it can be changed to horizontal by typing "h" inside the box displaying the attribute. Each menu is assigned an output token. On receiving this output token the run-time support module displays the menu. The run-time support module recognizes the menus by the output tokens, not by the menu names.

A variable number of menu items can be associated with a menu. Each menu item is assigned a unique input token name. When a menu item is selected the associated input token is generated.

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 a procedure written by the user interface designer. 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 hiererachy can be as complex as desired.

5.5.5. Input Token Definition

Input tokens can be associated with a window by selecting the "Input Token Definition" command from the ipes menu. 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 run-time support module receives the events generated as the result of user interactions with the user interface and generates the corresponding input token. Therefore, depending upon the event number more than one input token can be generated from the same window. The system allows the designer to delete or add any number of input tokens during a specification session.

5.5.6. Output Token Definition

Output tokens can be associated with a window by selecting the "Output Token Definition" command from the ipes 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. The display procedure can be chosen from the library of display procedures or it can be a procedure written by the interface designer. The display procedure associated with an output token can easily be changed by typing the new display procedure name. This facilitates the easy customization of user interfaces.

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.

5.5.7. 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 ipes 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 5.2.

To go to the previous level in the tree of windows the "Previous Level" command may be used. The system responds with the display of the windows defined at the root of the current sub-tree.

The Next and Previous Level commands can be used to traverse and modify the information stored at various nodes in the tree of windows. A complex hierarchy of windows can be easily created and maintained using these commands. The system

does not put any limit on the depth of the tree or number of windows at a particular level of the tree.

5.6. The ipes Editor

Ipes provides facilities for changing, inserting, appending, and deleting information associated with windows, menus, input tokens, and output tokens. These facilities can be used at the same time as the definition for window, menu, input token, and output token.

Except in the case of window definitions, ipes uses control characters to recognize the editing commands. Ipes recognizes Control-d as a delete command, Control-i as an insert command, and Control-a as an append command. In the case of window definitions, the editing commands are entered from the extra buttons on the cursor puck. Button PF2 is used for resizing a window, button PF3 is used for undoing the last corner entered, and button PF1 is used for deleting a window. The editing features supported by ipes are listed in Figure 5.3. The user manual describing in detail how to use ipes for specifying the design information appears in Appendix A2. The use of ipes for specifying the presentation component of an example graphical object editor is described in Appendix A3.

5.7. Help and Error Reporting

Ipes provides different help messages depending on the level of experience of the designer. For a new or less experienced designer, the help messages produced by ipes are quite detailed. For an experienced designer the help messages are terse. The generation of help messages can be turned on or off any time during the execution of ipes. The help feature is automatically turned on for the less experienced user on the invocation of ipes. The messages appear in the window across the top of the display screen.

Window Definitions
- Undo the last corner entered
- Remove a window
- Add a new window
- ReSize an existing window
- Change the window attributes

Menus
- Change the menu attributes
- Delete a menu
- Add a new menu

Menu items
- Change the menu item attributes
- Delete a menu item
- Insert a menu item before the current item
- Append a menu item after the current item
- Add a menu item at the end of the menu item list

Input Tokens
- Change the input token attributes
- Delete an input token
- Add an input token

Output Tokens
- Change the output token attributes
- Delete an output token
- Add an output token

Figure 5.3 Editing Features supported by ipes

Ipes detects and reports the errors committed by the designer while entering the design specifications. The error messages appear in same window as the help messages.

5.8. Novice and Expert Users of ipes

Ipes has two levels of use; "novice" and "expert". A novice may use only the basic set of commands. In this mode all the input is done through one button on the cursor puck, and all through the ipes session the designer is guided by the 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. An expert user of ipes is thus provided with a wider set of commands with fewer and terse prompting messages. A profile for each user of ipes (called "userprofile") is created and maintained by the program, initially tagging each user a novice. The userprofile file stores the status

("novice" or "expert") of the user, the number of times ipes invoked, and the number of times the designer sucessfully finished the specification sessions. A novice gets upgraded to an expert based on the number of successful executions of the ipes.

5.9. Output of ipes

The output produced by ipes is used to drive the run-time support module and provide interface for the dialogue control component and the application interface model in the user interface. 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 sections.

5.9.1. Data Base Description

The design information for the presentation component is stored in an FDB data base. For generating this data base ipes uses the schema given in Appendix A1. The basic organization of the data present in the data base is shown in Figure 5.4. An object in the data base is represented by a frame. A window frame points to the next window in 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. A window frame also stores the relevant attributes for the window. The first window frame in the data base is pointed to by a special frame.

A menu frame points to the next menu associated with the same window and stores other important attributes for the menu. It also points to the first menu item in the menu.

A menu item frame stores the text or the icon name labeling the menu item, the

location of the item in the menu, and the name of the input token that gets generated when the item is selected. It points to the next menu item belonging to the same menu.

An input token frame stores the name of the token, event number associated with the token, and pointer to the next input token associated with the same window.

An output token frame stores the name of the token, name of the display procedure associated with the token, and pointer to the next output token associated with the same window.

5.9.2. 'C' Procedures

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

5.9.3. 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 ipes 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.

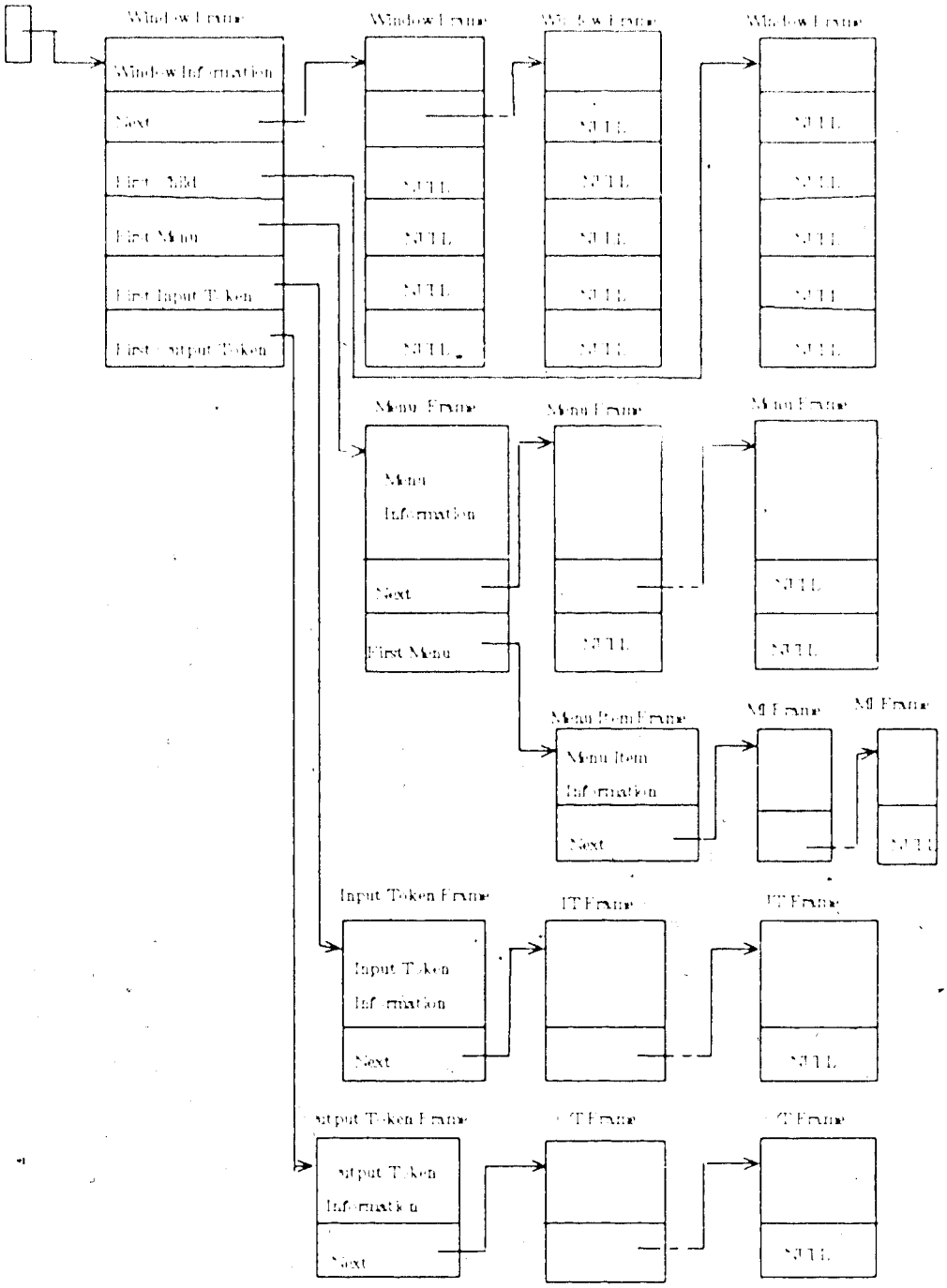

Figure 5.4 Organization of the Data Base

5.10. 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 peg is driven by the data base created by the specification program ipes. 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 ipes. This code is compiled and linked with the other run time routines. Peg 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 and the last part receives the output tokens and is responsible for their display. The functions performed by each of these parts are described in detail in the following sections.

5.10.1. Menu Display

This part of peg is responsible for the functions related to the display of menus and the associated bookkeeping. In order to handle all the menu related functions this part performs the following functions.

- Allocate space for each menu item.
- If the menu item is labeled as textual, display the text after centering it horizontally as well as vertically.
- If the menu item happens be an icon, make a call to the associated procedure and pass the appropriate parameters.
- Keep track of the current menu selection and highlight the selected menu item.
- On selection of a menu item generate an event so that the appropriate input

token may be generated.

For allocating space to the menu items the program checks for the allocation option. If the allocation was done by the designer then it simply allocates the space and position to each menu item as specified by the designer. Otherwise, peg divides the whole menu area equally for each menu item and allocates the space in the order of their appearance in the list of menu items.

5.10.2. Generating Input Tokens

Peg receives all the input events generated as a result of the user's interactions with the user interface and generates the input tokens. Depending upon the event number and the window from where the events originate the program performs a table look up to determine the input token number. The information is then restructured in the form of an input token and sent to the dialogue control component.

5.10.3. Generating Images

Peg receives output tokens from the other parts of the program and generates images corresponding to these tokens. To be able to display an output token peg finds the window address where the token should be displayed and the display procedure responsible for displaying the token. To perform these functions it uses a table look up mechanism. After determining the required information a call is made to the display procedure with the output token and the window address as the parameters. The display procedure then generates the image in the specified window according to the information present in the output token.

5.11. Run-Time Errors

The run-time support module detects errors such as illegal event type, creating an already existing window, and illegal output tokens. The errors detected by the run-time support module are written into a file called "pcg.err". The system ignores the action in the cases of errors and continues the support for the user interface.

Chapter 6

Contributions and Directions for Future Research

6.1. Contributions

Although important works on user interfaces date as far back as the late 1960's [Foley1984], most of the ideas of User Interface Management Systems and automatic generation of user interfaces are comparatively new [Newman1968], [Kasik1982], [Guest1982], [Buxton1983], [Jacob1983], [Olsen Jr.1983a]. The majority of existing systems, with the exception of U of T-UIMS have concentrated largely on the dialogue control component with little or no attention devoted to the presentation component. In his paper "Automatic Generation of Interactive Systems", Dan Olsen noted that:

> "... we have relied heavily on concepts derived from work in compilers and programming languages. The reason for this is that any interactive system is characterized by its command language ..." [Olsen Jr.1983b].

In this thesis attention is focussed on the issues involved in automatic generation of presentation component of user interfaces. The model used as the basis of this work provides a top-down structured approach to user interface design. The designer concentrates on one component of the user interface at a time. 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 thesis is the illustration 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, as shown in the implementation, 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 thesis is to illustrate 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.

The catalogue of interaction techniques describes the contents of the library of interaction techniques. The catalogue helps the designer of graphics systems in selecting interaction techniques that best match both the application requirements and the user characteristics. A classification of interaction techniques based on the interaction task performed, their dimensionality, the type of feedback produced, and the virtual device used for implementing the technique has been proposed.

6.2. Directions for Improvements and Future Research

The input and output facilities provided in the presentation component can be extended to include sound, touch, and movement. Another extension could be to automatically adjust the size of the windows depending upon the amount of information to be displayed. More facilities can be added to the system to provide a better overview of the system being designed. The system could be extended to display the complete hierarchy of windows and allow the designer to jump more than one level at a time in the hierarchy.

A different direction for research could be to make the presentation component adaptive. Our system is capable of collecting data about various menu selections and the movement of graphical cursor on the screen. Depending upon the statistics generated from this data the system could adapt itself for a better performance or produce the results and let the designer decide about the action. Some work in this direction has been done by Edmonds [Edmonds1981].

References

Blesser1982.

T. Blesser and J. D. Foley, " Towards Specifying and Evaluating the Human Factors of User-Computer Interfaces", *Conf. on Human Factors in Computer Systems*, Mar. 1982.

Boehm1976.

B. W. Boehm, J. R. Brown and M. Lipow, " Quantitative Evaluation of Software Quality", *Proc. IEEE-ACM 2nd International Conference on Software Engineering*, 1976.

Buxton1983.

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

Cheriton1976.

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

Chia1985.

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.

Edmonds1981.

E. A. Edmonds, " Adaptive Man-Computer Interfaces", in *Computing Skills and the User Interfaces*, Academic Press, London, 1981, pp 389-426.

Edmonds1982.

E. A. Edmonds, " The Man-Computer Interface: A Note on Concepts and Design", *Int. J. Man-Machine Studies 16*, (1982), pp 231-236.

Feldman1982.

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, pp 111-116.

Foley1974.

J. D. Foley and V. L. Wallace, " The Art of Natural Graphic Man-Machine Conversation", *Proc. IEEE*, Apr. 1974, pp 426-471.

Foley1981.

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.

Foley1984.

J. D. Foley, V. L. Wallace and P. Chan, " The Human Factors of Computer Graphics Interaction Techniques", *IEEE-Computer Graphics and Applications 4*, 11 (Nov. 1984), pp 13-48.

Green1982.

M. Green, " *A Specification Language and Design Notation for Graphical User Interfaces*", Tech. Rep. 81-CS-09, Unit for Computer Science, McMaster University, Hamilton, Ontario, Canada, June 1982.

Green1983a.

M. Green, " A Catalogue of Graphical Interaction Techniques", *Computer Graphics*, Jan. 1983, pp 46-52.

Green1983b.

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

Green1984a.

M. Green, "Design Notations and User Interface Management Systems", *Proc. Seeheim workshop on User Interface Management Systems*, 1984.

Green1984b.

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

Green1984c.

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

Green1985.

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

Guest1982.

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

Jacob1983.

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

Kasik1982.

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

Krammer1980.

G. Krammer, "On Computer Problem-Solving Interaction", in *Methodology of Interaction*, Amesterdam, North-Holland, 1980, pp 272-292.

Lau1985.

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. (expected).

Murch1984.

G. M. Murch, " Physiological Principles for the Effective Use of Color", *IEEE-Computer Graphics and Applications 4*, 11 (Nov. 1984), pp 49-53.

Newman1968.

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

Olsen Jr.1983a.

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

Olsen Jr.1983b.

Dan R. Olsen Jr., " Automatic Generation of Interactive Systems", *Computer Graphics*, Jan. 1983, pp 53-57.

Parnas1972.

D. L. Parnas, " A Technique for Software Module Specification with Examples", *Comm. ACM 15*, 5 (1972), .

Roach1982.

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, pp 102-105.

Seattle1983.

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

Appendix A1

Database Schema

```
FRAME head
    first_window = -1;
END

FRAME wind_list META
    fheader = 1;
    name ;
    minx = 0.0;
    maxx = 1.0;
    miny = 0.0;
    maxy = 1.0;
    llx; lly;
    urx; ury;
    bg_color = 0;
    dr_color = 1;
    bnd_color = 1;
    intt_tech;
    intt_call;
    window_token;
    parent_frame = -1;
    first_child = -1;
    first_menu=-1 ;
    first_in_tok = -1;
    first_out_tok = -1;
    next= -1;
END

FRAME input_token META
    fheader = 2;
    inname;
    intype;
    innext = -1;
END

FRAME  output_token META
    fheader = 3;
    outname;
    outproc;
    outwindptr;
    outnext = -1;
END

FRAME  menu_header META
    fheader = 4;
    menu_name;
    menu_type;
    orientation;
```

```
        menu_token;
        coord_choice;
        fst_menu_item = -1;
        next_menu = -1;
    END

    FRAME  menu_item META
        fheader = 5;
        in_tok_name;
        in_tok_no;
        item_type;
        boxllx; boxlly;
        boxurx; boxury;
        linx;
        liny;
        line;
        iconname;
        next_item = -1;
    END
```

Appendix A2

User Manual

There are two steps involved in the complete design of the presentation component of a user interface. 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 "ipes" (interactive presentation component specification), accepts the design specifications from the designer and generates an FDB data base, token tables, and 'C' procedures. The second part, called "peg" (presentation component generation), consists of a number of procedures which provide the run-time support for the presentation component of the user interface. This part is driven by the data base and the token tables. The 'C' procedures produced by ipes are compiled and linked with peg procedures. The entire sequence of creating a presentation component is shown in Figure A2.1.

1. The Specification Step

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

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

The interactive specification program "ipes" is used to enter the design information. The ipes screen is divided into four areas as shown in Figure A2.2. The

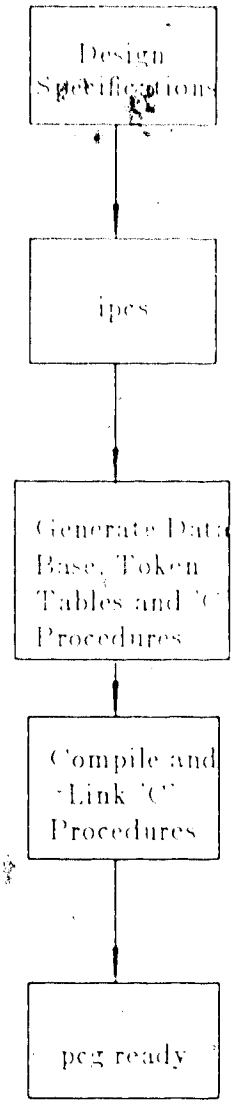

Figure A2.1 A Sequence for Constructing a Presentation Component

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 ipes 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 ipes is presented.

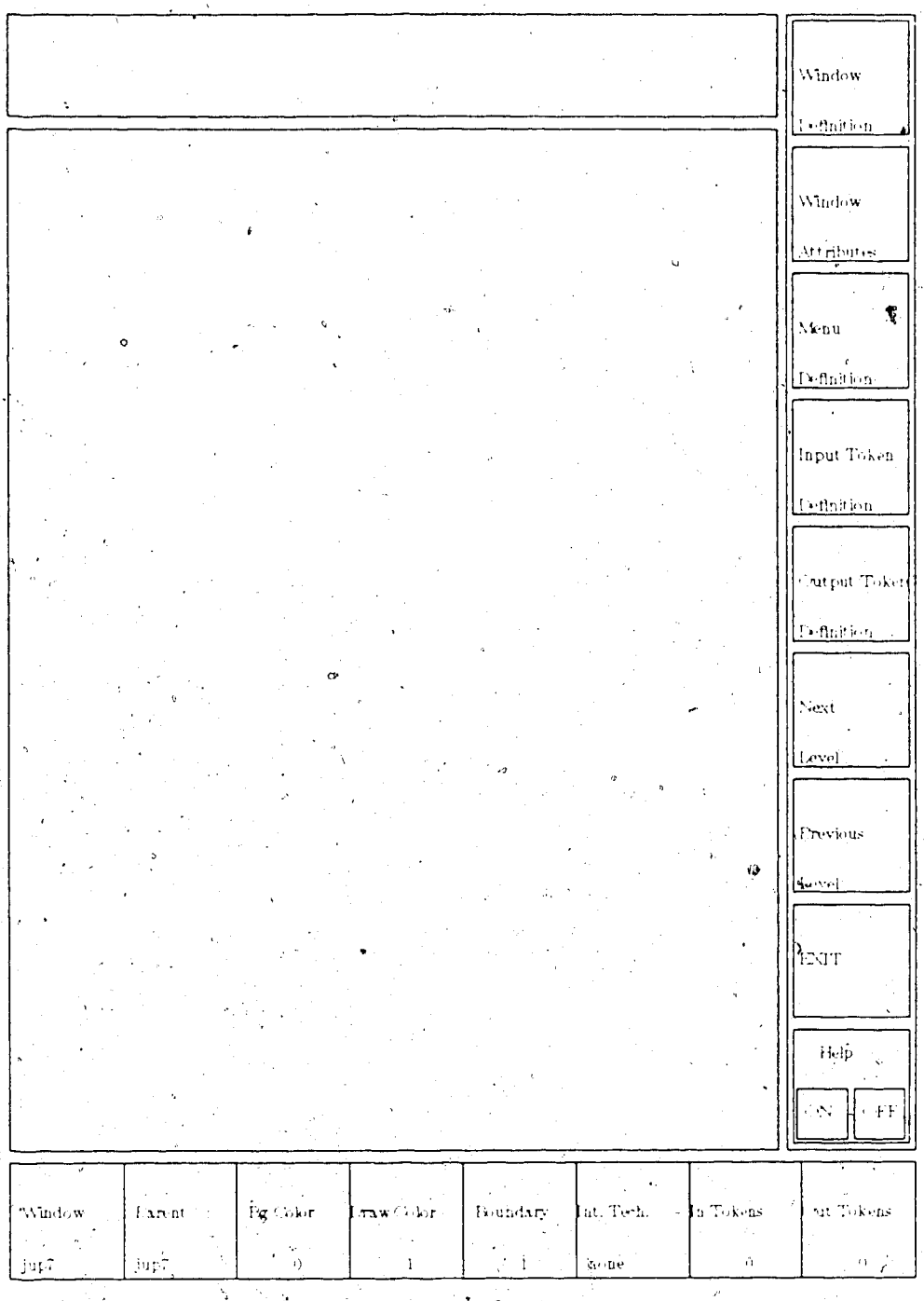

Figure A2.2 The ipcs Screen Layout

2. Running ipcs

Ipcs stores the design information entered by the designer in an FDB database. This database should be initialized before starting a specification session. The schema for the database is defined in the file called "the.schema" in singh/new/code

directory. The following command may be used to initialize the database:

schema <dbname> the.schema

The "dbname" is the name of the database where the design information is stored. This name should be entered when ipes prompts for the name of the database. The following command should be used to invoke the program:

ipes

3. Window Definition

The "Window Definition" command is used for creating windows in the work area. This command can also be used for deleting or resizing existing windows. To create a window the user is expected to point at the two opposing corners of the window. This can be done by moving the graphical cursor to the desired positions and hitting PF1. A window can be deleted by moving the graphical cursor inside the window and hitting PF1. To resize an existing window move the graphical cursor inside the window close to the lower left or upper right corner and hit PF2. The input now changes to sample mode and the selected corner can be dragged to the desired position. The corner can be fixed by hitting PF1. While creating windows if one of the corners (the first corner for any window) is to be undone, hit PF3.

4. Window Attributes

The "Window Attribute" command is used for assigning attributes to the windows. This command assigns some default attributes to each of the windows. The default attributes for a window are window name, window limits, background colour, drawing colour, and boundary colour. An attribute can be changed by moving the graphical cursor on top of the box displaying the attribute and typing the new value.

To associate an interaction technique with a window the designer is required to type the name of the interaction technique along with its parameters. The call to the interaction technique should be typed exactly as it would be typed in a 'C' program. There should not be any ';' at the end of the call. In this call the name of the window should be "wind", irrespective of the name of the window to which the interaction technique is associated. All the events generated by the interaction technique should be sent to window named "term". Any event type less than 512 can be generated by the interaction technique. For example, to associate a graphical potentiometer (g_pot) to a window called "w11" the call should be typed as follows:

g_pot(wind, other_parameters, term, 1)

In a normal case, depending upon the size of the window, one, five, or ten attributes are displayed at a time. In the case of five or one attributes, the next set of attributes can be displayed by moving the graphical cursor inside the window and hitting carriage return. If the size of a window is smaller, the system cannot display any attributes unless it is changed. To temporarily adjust the size of such a window the designer is required to move the graphical cursor inside the window and hit PF2. In the case of overlapping windows, the attributes of one the windows may overlap with another window. To flip the windows in such a case move the graphical cursor inside the window to be displayed and hit PF3. The selected window temporarily becomes the top most window. The screen can be redrawn in its original form by hitting PF1 any where in the work area.

5. Menu Definition

This command is used for defining menus and associating them with windows. To associate a menu with a window select the window after the selection of the "Menu Definition" command. A window can be selected by moving the graphical cursor inside

the window boundary and hitting PF1.

A menu definition consists of two parts; menu header definition, and menu item definition. The menu header information consists of information relevant to the whole menu. It affects the appearance and location of the menu. This part of the menu definition consists of menu name, menu type, menu items placement option, menu orientation, and name of the output token associated with menu. Some of the menu header attributes are assigned default values. The default value for the menu type is "fixed", it can be changed to "pop-up" by typing "p" inside the box. The default menu items placement option is "system", it can be changed to "designer" by typing "d" inside the box. The default menu orientation is "vertical", it can be changed to "horizontal" by typing "h" inside the box. The menu name and output token name are entered by the designer. These names should be unique. After the menu header definition is complete, the menu items definition can be started by hitting a carriage return inside the window.

The menu item definition consists of the input token name and its type. The type of a menu item can be textual or iconic. The default type is "textual"; it can be changed to iconic by typing "i" inside the box for the item type. If the menu items placement option in the menu header is "designer", then the lower left and upper right corner coordinates for the area allocated to the menu item are entered by the designer. In the case of textual type menu items the designer can enter up to three lines of text. Each of these lines can be a maximum of 80 characters long. In the case of an iconic menu item the designer is required to enter the name of the procedure drawing the icon. No parameters for the procedures are required. The text or icon names are entered after entering the input token name, menu item type, the coordinates of the box, and hitting the carriage return. The next item in the menu can be defined by hitting the carriage return. The process can be repeated to assign more than one item to a menu.

A menu item may be deleted by hitting Control-d when the item is visible. This will delete the displayed item from the list and display the next item. If the deleted item happened to be the last item, the next item is empty. To insert a menu item before the current item enter a Control-i. The system will display an empty item. The designer can define its contents. To add a menu item after the current item enter Control-a. This will display an empty menu item whose contents can be defined by the designer.

More than one menu can be associated with a window by typing Control-a when the menu header is visible. The process of defining the contents of the menu remains the same as described earlier. To delete a menu enter Control-d when the menu header is displayed.

6. Input Token Definition

Any number of input tokens can be associated with a window by selecting the "Input Token Definition" command. The window can be selected by hitting PF1 inside the boundary of the window. An input token definition consists of an event type and input token name. The event type is the type of event which triggers the token. The token names must be unique. To define the next input token hit the carriage return after the definition of one of the tokens is complete.

An input token can be deleted by hitting Control-d when the token is displayed.

7. Output Token Definition

Any number of output tokens can be associated with a window by selecting the "Output Token Definition" command. The window can be selected by hitting PF1 inside the boundary of the window. An output token definition consists of output token name and the associated display procedure name. The output token names are eight characters long and must be unique. A display procedure can be associated with

an output token by typing its name. No parameters for the procedure are required. To define the next output token hit the carriage return after the definition of one of the tokens is complete.

An output token can be deleted by hitting Control-d when the token is displayed.

8. Next Level Definition

A tree of windows can be created by using the "Next Level" and "Window Definition"/"Window Attributes" commands from the ipes 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 then corresponds to the selected window and child windows can be created.

9. Previous Level

To go to the previous level in the tree of windows the "Previous Level" command may be used. The system responds with the display of the windows and their attributes defined in the previous level.

10. Help and Error Reporting

Ipes provides different help messages depending on the level of experience of the designer. For a new or less experienced designer, the help messages produced by ipes are quite detailed. For an experienced designer the help messages are terse. The generation of help messages can be turned on or off any time during the execution of ipes. The help feature is automatically turned on for the less experienced user on the invocation of ipes. The messages appear in the window across the top of the display screen.

Ipes detects and reports the errors committed by the designer while entering the design specifications. The error messages appear in the same window as the help messages.

11. 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 all the input is done through one button on the cursor puck, and all through the ipcs session the designer is guided by the 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. An expert user of ipcs is thus provided with a wider set of commands with fewer and terse prompting messages. A profile for each user of ipcs (called "userprofile") is created and maintained by the program, initially tagging each user a novice. The userprofile stores the status ("novice" or "expert") of the designer, the number of times ipcs has been invoked, and the number of times the designer sucessfully finished the specification sessions. A novice gets upgraded to an expert based on the number of successful executions of ipcs.

12. Output of Ipcs

The output produced by ipcs is used for driving the run-time support module and provides the interface to the dialogue control component and application interface model. The information produced for the run-time support module consists of an FDB data base and 'C' procedures whereas the information for the dialogue control component and application interface model consists of tables of input and output tokens. The complete sequence of creating a presentation component and the flow of data is shown in Figure A2.3.

12.1. 'C' Procedures

The 'C' procedures generated by ipcs are mainly used for passing the parameters to the called interaction techniques and display procedures. The 'C' code also helps in loading the appropriate procedures from the library of interaction techniques, library

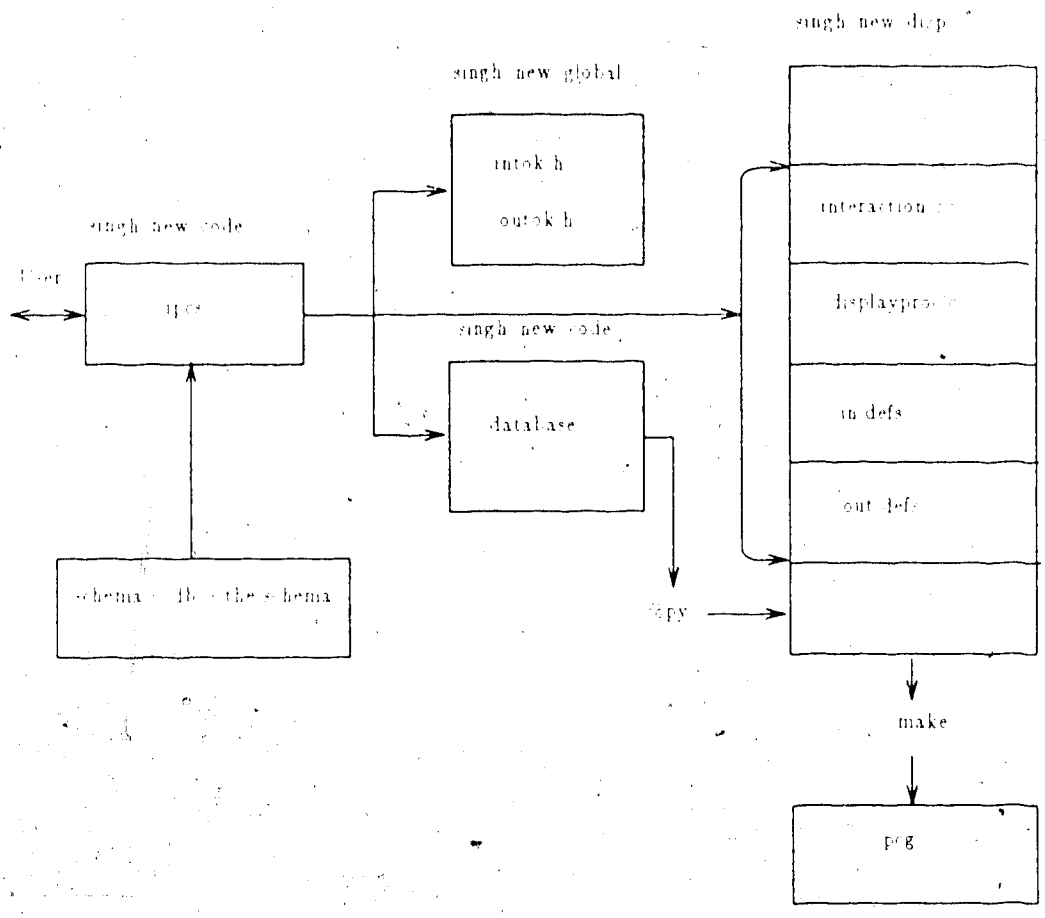

Figure A2.3 Ipcs and its Output

of display procedures, and library of icons. These procedures are compiled and linked with the other routines for the run time support.

12.2. File of Input/Output Tokens

This file contains the names of all the input and output tokens along with token numbers. These numbers are assigned by the 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.

13. Run-Time Support Module

The run-time support for the presentation component of the UIMS is provided by the routines in "pcg". The routines to provide the run-time support are located in

singh/new/disp directory. Peg is driven by the data base and the token tables created by the specification program ipes. The database files should be copied from singh/new/code to singh/new/disp. Peg retrieves the information from the database and restructures the information as required. It also gets some help from the 'C' code generated by the ipes. This code is compiled and linked with the other run time routines. Peg 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 and the last part receives the output tokens and is responsible for their display.

14. Files to be Compiled and Loaded

The files interaction.c and displayproc.c have to be compiled and linked with the run-time support module. The files are created in singh/new/disp directory. To do so run "make" located in singh/new/disp directory.

15. Files to be Included

The files out.defs and in.defs are read by the run-time module to gather information for input and output token definitions. These files are different from intok.h and outok.h which are created for the dialogue control component and application interface model. The user is not supposed to do anything about this step. These files are created in singh/new/disp directory.

16. Files for Other Components

The files outtok.h and intok.h are used by the other components. These files are created in singh/new/global directory.

17. Run-Time Errors

The run-time support module detects errors such as illegal event type, creating

an already existing window, and illegal output tokens. The errors detected by the run-time support module are written into a file called "peg.err". The system ignores the action in the cases of errors and continues the support for the user interface.

Appendix A3

An Example

The use of ipes to enter design specifications for the presentation component of a user interface is illustrated with the help of an example to create a menu driven graphical object editor. This graphical editor can be used for adding, moving, or deleting graphical objects of variable sizes on the screen. The graphical objects considered in this example are circle, square, and triangle. The size of each of these objects can be controlled by a graphical potentiometer. The screen layout for this editor consists of three windows as shown in Figure A3.1. The main window is called the work window where all the objects are positioned. The menu window is located on the right side of the display screen. The third window, called the potentiometer window, is located across the bottom of the screen and is used for displaying the graphical potentiometer. The menu consists of a mixture of iconic and textual menu items. It contains the three graphical objects and three commands, namely, "Move", "Delete", and "Exit", displayed as text.

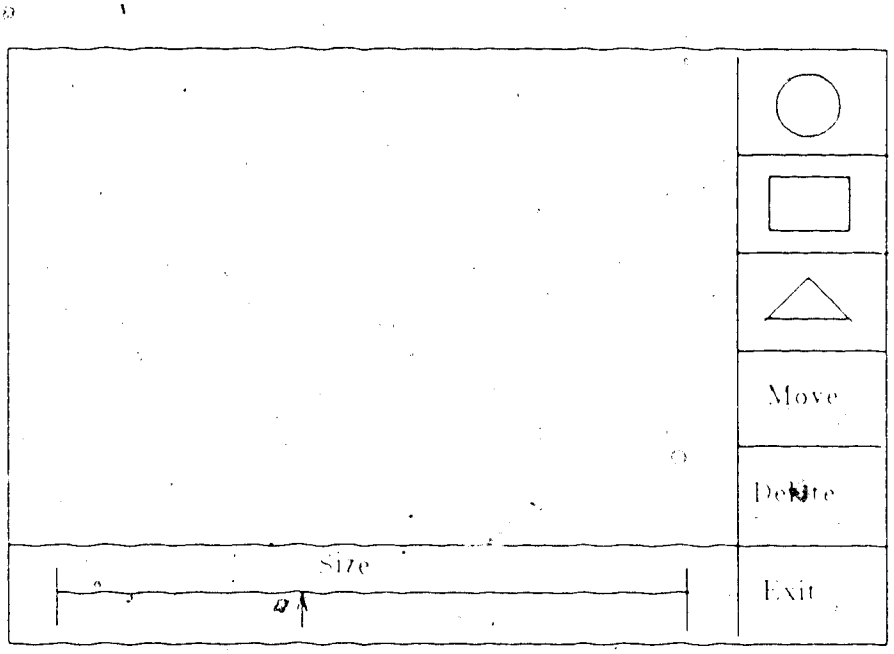

Figure A3.1 The Screen Layout for the Graphical Editor

To position a graphical object in the work area the user selects one of the circle, square, or triangle from the menu and points at the position where the object should be located. The size of an object can be controlled by the value of the graphical potentiometer. The command selection is open ended, that is, the selected command remains active as long as the user does not change it explicitly. To change the position of an object the "Move" command is used. The user first points at the object to be moved and then at the new position. To delete an object from the screen the "Delete" command is used. The object can be deleted by pointing at it after the comand is selected. It may be noted that the value of the potentiometer does not affect the move or delete operations. To exit from the editor the "Exit" command is used.

1. The Design

The work window in the editor needs to generate one input token. This token is used for passing the corrdinates of the point in the work area to the dialogue control component. The token can be generated by an event of type "1" in the work window. To be able to draw and erase three types of objects this window needs six output tokens. Three of these tokens are used for drawing the objects and the other three for erasing them. Each output token has an associated display procedure to perform the required function. The menu window generates one input token for each command in the menu. This window does not have any output token associated with it. The window for the graphical potentiometer has one input token associated with it. This input token is used for changing the size of the graphical objects. This window does not have any output token associated with it.

2. The Specification Step

The first step in the specification process is to initialize the design database. The database for the example editor is called "editor". This database can be initialized by the following command

schema editor the schema

The program "ipes" is invoked by typing its name. The name of the database, "editor", is entered when ipes prompts for the database name. The three windows, as shown in Figure A3.1, can be created by using the "Window Definition" command from the ipes menu and pointing at their opposing corners. After creating the windows the ipes screen looks as shown in Figure A3.2.

To associate attributes with the windows, "Window Attributes" is selected from the ipes menu. On selecting this command the display screen is redrawn by the program. This command also assigns default attributes to each of the windows. For the example editor, the window output token name is to be associated with each of the windows. This can be done by moving the graphical cursor on top of the box for the output token and typing its name. The window output token name can be the same as the window name. Since only one attribute is displayed at a time in the case of the potentiometer window, the Carriage Return (CR) is entered till the required attribute is displayed. The graphical potentiometer interaction technique is to be associated with the potentiometer window. This can be done by typing the following call in the box for interaction technique.

gpot(wind, HORZS, 0.0, 1.0, "Size", term, 1)

After this step the ipes screen looks as shown in Figure A3.3.

To associate the menu with the menu window, the "Menu Definition" command is selected. After selecting this command the designer points at the menu window. Ipes then displays the menu header in the menu window with the default attributes. The designer needs to type in the menu name and the menu output token name. This can be done by typing "menu" in the boxes for the attributes. The display on the screen looks as shown in Figure A3.4.

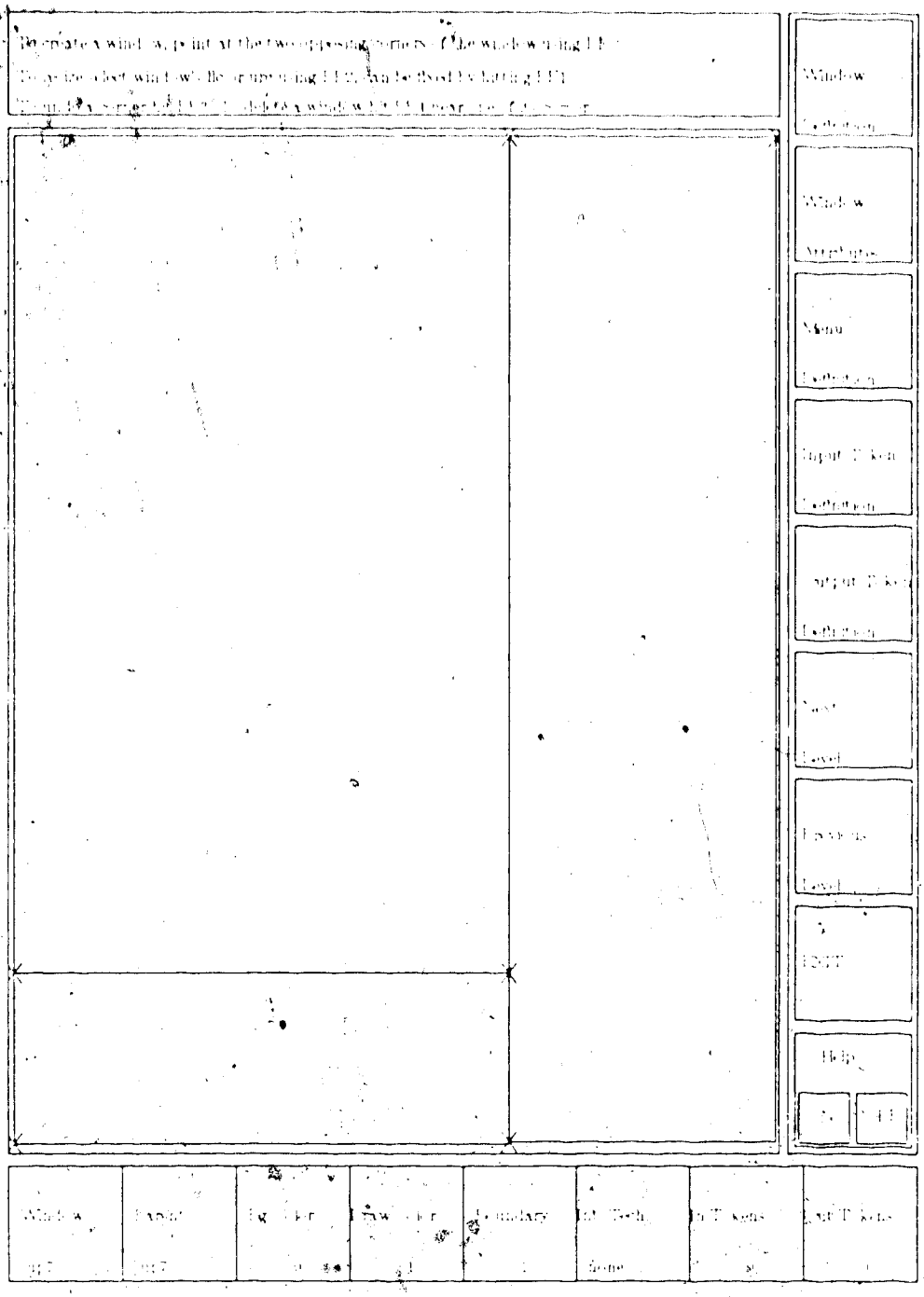

Figure A3.2 Ipes screen Layout after Creating Windows

The menu items for the menu can be defined by hitting a Carriage Return in the menu window. For each menu item the designer needs to enter the input token name and the text or the name of the icon labelling the item. For iconic items the designer needs to change the default item type ("text") to iconic by typing "i" in the box for

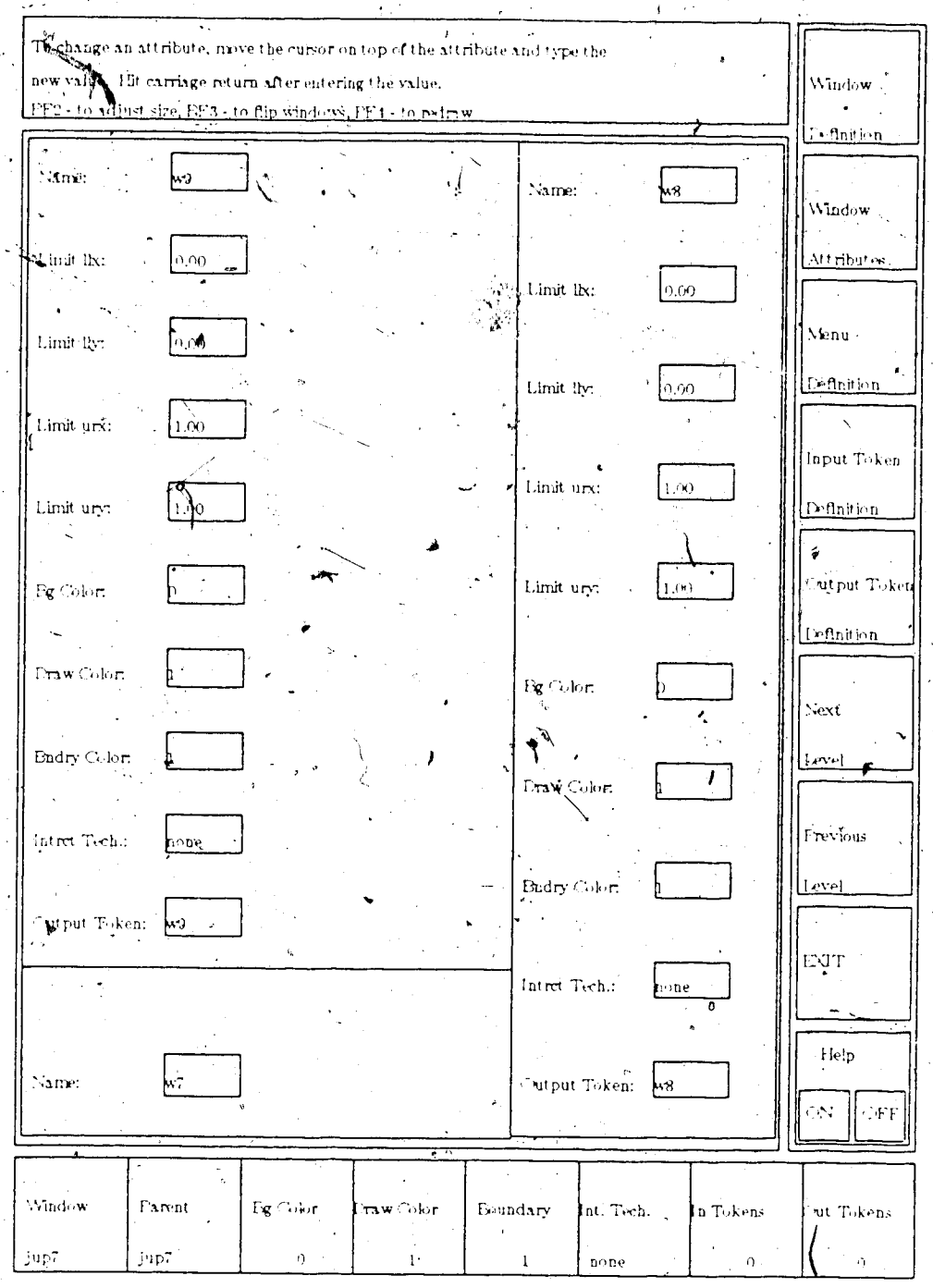

Figure A3.3 Ipcs Screen Layout after associating Window Attributes

the menu item type. In the next step the program then asks for the name of the procedure drawing the icon. For example, in the case of the circle, the name of the input token can be "circle" and the name of the procedure drawing the circle can also be "circle". In the case of textual items, the program provides boxes for entering the

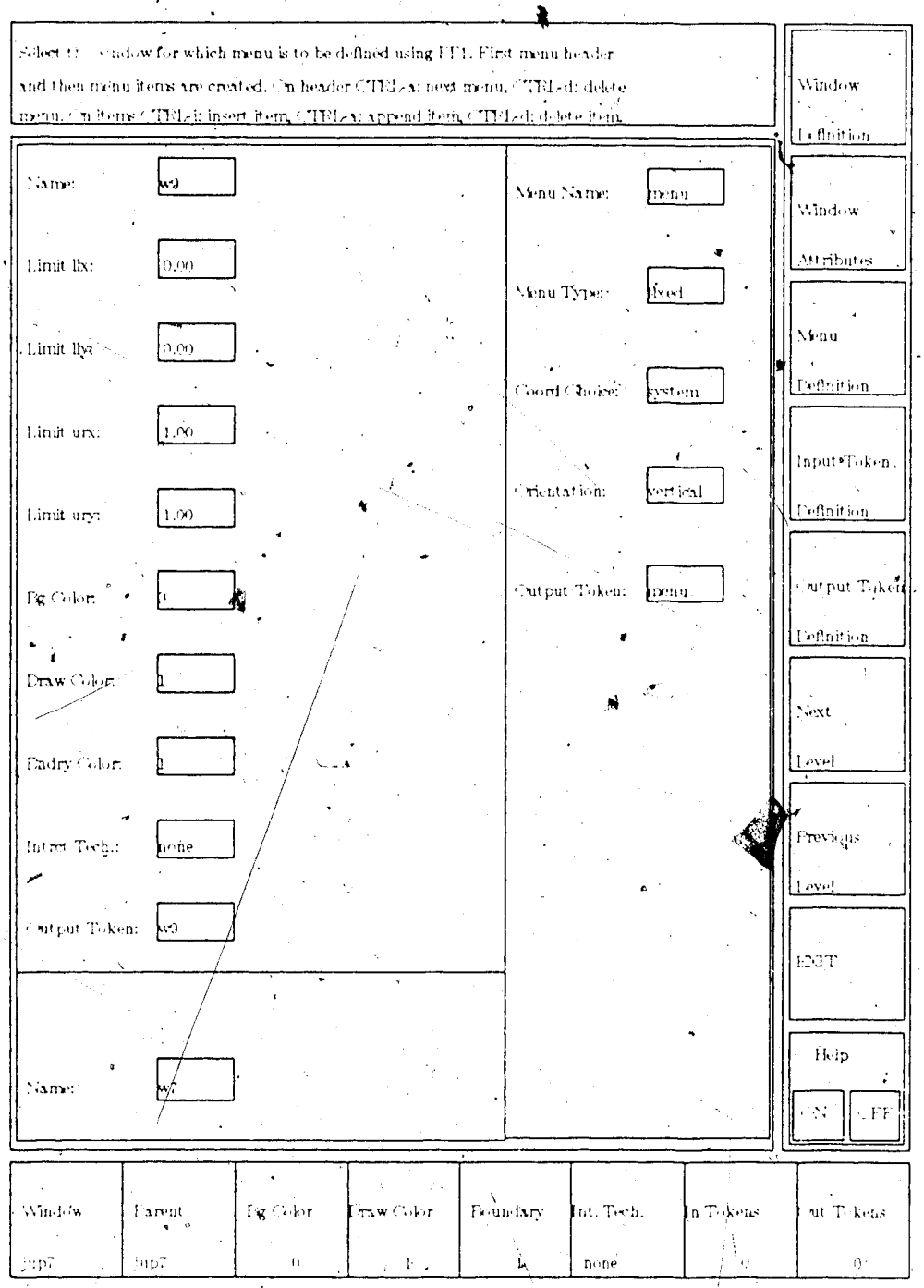

Figure A3.4 Ipes Screen Layout after defining Menu Header

text to be displayed in the menu item. In this example, "Move", "Delete", and "Exit" are entered for the fourth, fifth, and sixth menu items respectively. This completes the definition of the menu.

In the next step, input tokens are associated with the work and potentiometer windows. This is accomplished by selecting the "Input Token Definition" command from the ipcs menu and pointing at the required windows. For each input token the input token name and the associated event type are entered. In the case of work window, the name of the input token is "point". The default type of the event need not be changed in this case. Similarly input token name "size" can be associated with the potentiometer window. The screen in this case looks as shown in Figure A3.5.

The last step in the specification sequence is the definition of the output tokens. This can be started by selecting the "Output Token Definition" command from the ipcs menu and pointing at the work window. For each output token the output token name and the name of the associated display procedure is entered. For example, the output token for drawing the circle has token name "dcirc" and the name of the display procedure is "drawcirc". The other output token names are "dsq", "dtri", "ecirc", "esq", and "etri". The names of the display procedures are "drawsq", "drawtri", "erasecirc", "erassq", and "erastri" respectively. The screen layout in this case looks as shown in Figure A3.6. This completes the specification required for the example editor.

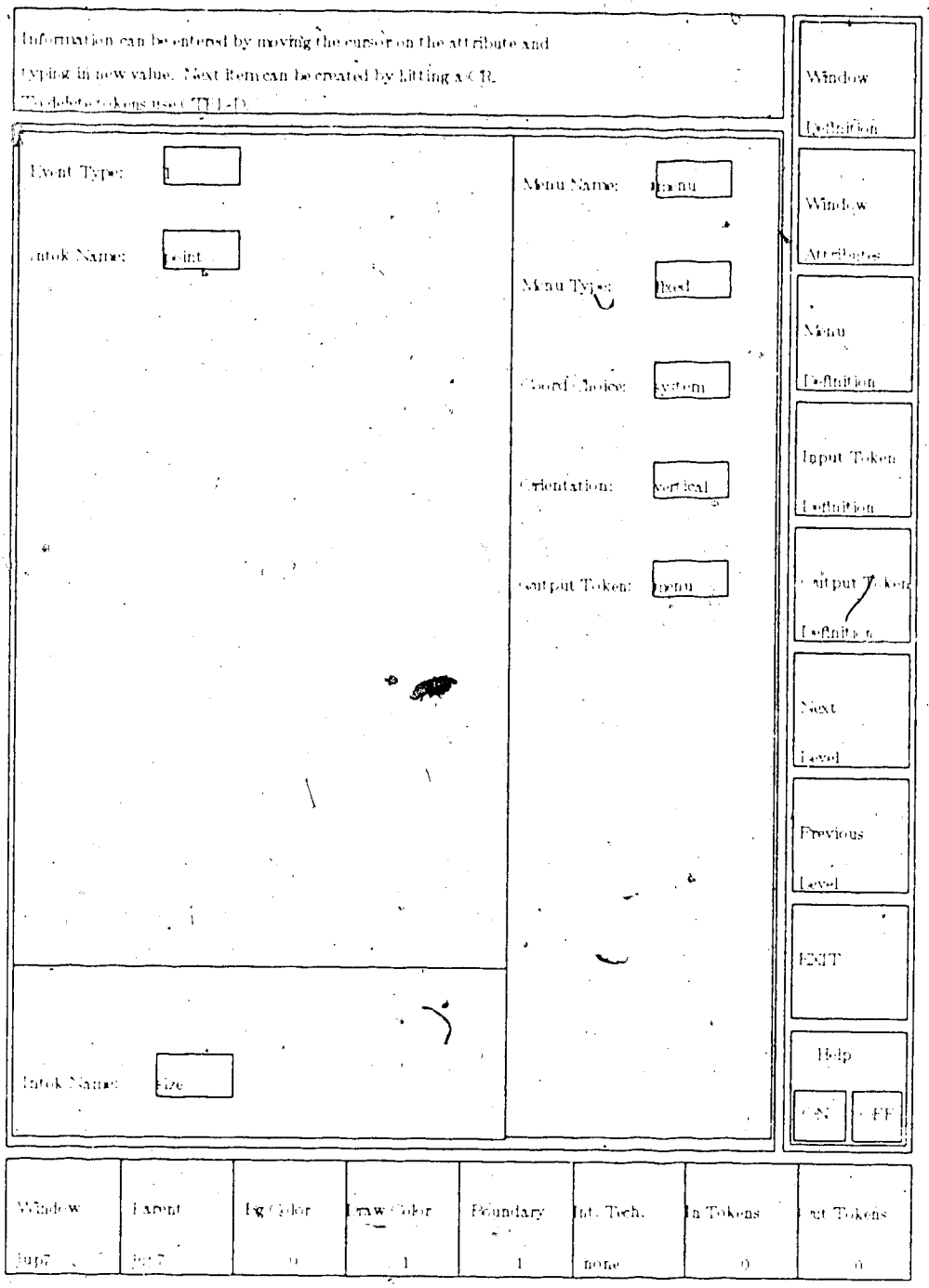

Figure A3.5 Ipes Screen Layout after defining Input Tokens.

The specification session can be ended by selecting the "Exit" command from the ipes menu. On selecting this command, ipes creates the database, token tables, and files of 'C' procedures. The files of 'C' procedures are compiled and loaded with the routines in the run-time support module.

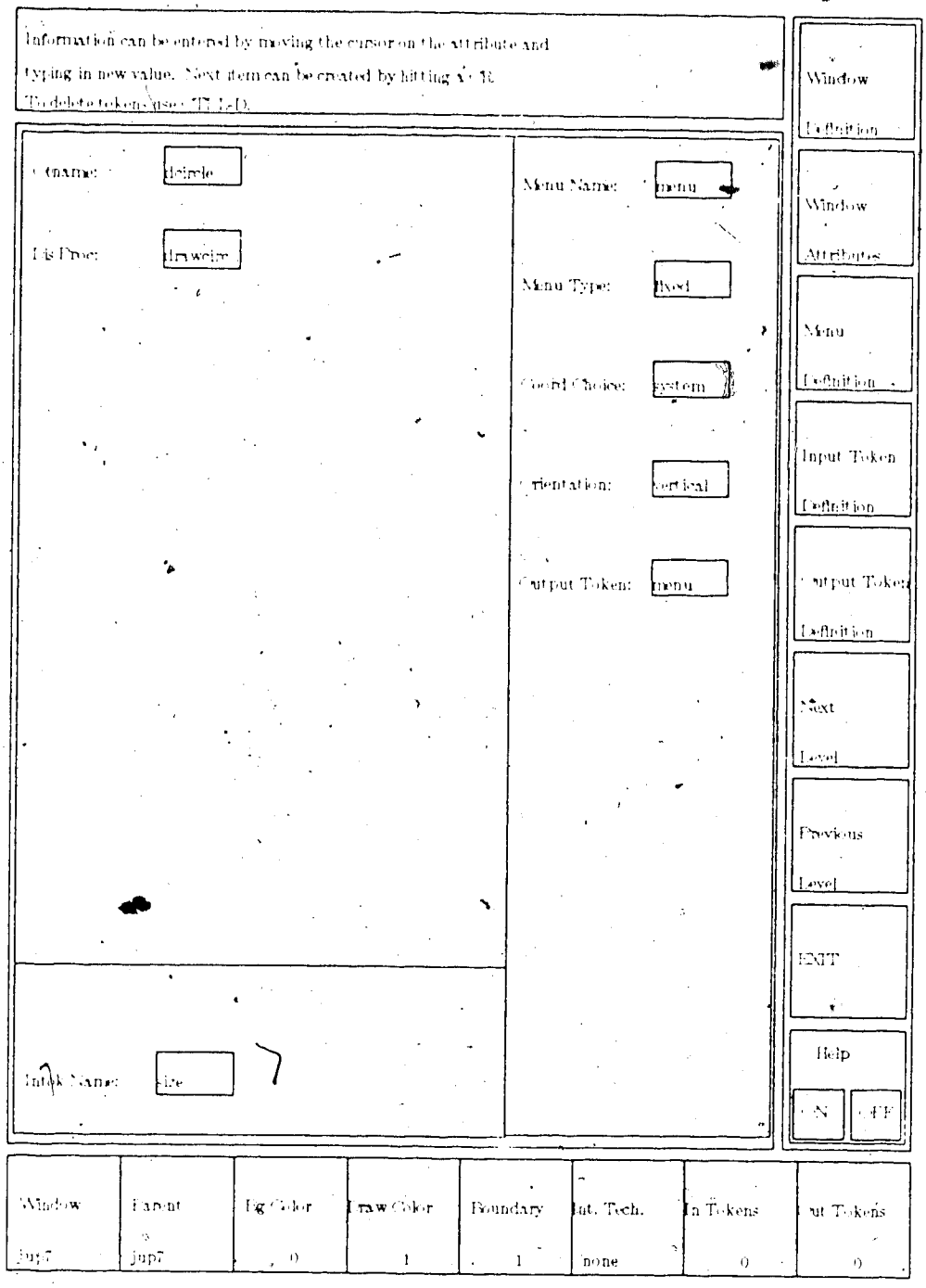

Figure A3.8 Screen Layout after defining Output Tokens