1986

GUSIP: Graphical User Interface Genreator for Application Programs

Mohamad A. Alda'Da'
University of Rhode Island

GUSIP: GRAPHICAL USER INTERFACE GENERATOR FOR APPLICATION PROGRAMS

BY

MOHAMAD A. ALDA'DA'

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

1986

MASTER OF SCIENCE THESIS

OF

MOHAMAD AHMAD ALDA'DA'

Approved:

Thesis Committee

Major Professor ----------

Dean of the Graduate School

UNIVERSITY OF RHODE ISLAND

1986

A C K N O W L E D G E M E N T

I acknowledge with thanks and gratitude the support of Dr. Leonard Bass whose guidence, suggestions and continuous followup were the main reasons behind the success of this project. I also acknowledge the great efforts of my wife, Amal, who has suffered with me through sleeples nights, helped me in typing the draft and final copies of the thesis and without whose encouragement and enthusiasm this would not have been possible. To my friends, the faculty, graduate students and staff of the computer science department, at URI, and all those who have participated, one way or the other, in making this possible I express my sincere thanks for their appreciated efforts. However, any descrepencies or mistakes that might appear are my own fault.

ABSTRACT

Human computer interaction is in the early stages of development as a science. Although the field contains principles that are sometimes contradictory, a concensus is emerging about the separation of the functionality of a computer system (i.e applications) from the user interface. This separation frees the application programmer from low level details, so as to be able to concentrate on higher application specific aspects of the user interface.

GUSIP (Graphical User Interface Generator for Application Programs) is an application support system which deals with the generation of graphical user interfaces and allows the separation of an application from its graphical user interface. The graphical user interface can thus be designed independently of the main application. GUSIP also facilitates the tailoring of the graphical user interface of any application to its user needs by providing a graphics editor which allows the interface designer to design (draw) the screens using graphical editing facilities (drawing of geometric shapes, free form drawing, moving, copying, rotating, deleting, scaling ...etc). The drawn graphical user interface is linked to

the application by means of a run time module which handles the output of the editor and allows passing graphical data between the application and the display system. The combination of these two modules allows the design of several graphical user interfaces for an application, and the modification of an existing graphical user interface. The significance of GUSIP is that the above operations can be done without involving the application program, and hence the application programmers do not have to deal with tasks that are irrelevant to the functionalities of their application.

TABLE OF CONTENTS

INTRODUCTION

Computer hardware is becoming cheaper, and a great deal of emphasis is being placed on making software systems interact directly with human users. A major stumbling block, however, is the high cost of producing a quality user interface. This is because current understanding of human-computer interaction is extremely limited. After designing a user interface, one cannot know how it will perform. Therefore, one must accept, at the outset , the inevitable intertwining of specification, design and implementation. The design of the user interface then becomes an iterative process, with each iteration consisting of the three phases of design, implementation and evaluation. The introduction of graphics has complicated matters even more, graphic displays have become part of every computer system, which made the graphical user interface an essential part of any system intended to interact directly with users.

Traditionally, interactive graphical applications have been written using conventional programming languages, low level tools and often ad hoc techniques.

The cost of doing so has been time, frustration, and quality of the end product. The motivation to develop improved tools for designing a graphical user interface can thus be seen as a desire to increase the number of iterations one can afford to pass through the loop of designing a graphical interface, and at the same time decrease the time required for doing so.

It has been the goal of many research projects to produce tools which will automate the production of graphical user interfaces and thus make the production of quality graphical interfaces more economical. The result of those were systems that aim at making available to the interface designer tools which allow the design of a graphical user interface, that is, independent, at least to some extent, from the application program. Many of the tools developed, have been either too complicated for an interface designer to use, or too simple and with limited capabilities to be of any use in a sophisticated application, and in general these are not well suited to be used with different types of applications.

SYNGRAPH (Syntax Directed Graphics) [8, 9], is a user interface generator for interactive graphics systems. The system applies the principles of syntax analysis, parser generation and data abstraction to the development of man machine interfaces. To provide the semantics of interaction, the application supplies a set of data types and a set of procedures that manipulate these types. The data types define pictures to be displayed as well as parameters to the procedures which are invoked as semantic actions of the interaction input parser. The interaction generator produces PASCAL source code for a table driven recursive descent parser, which allows attributes to be implemented as parameters and local variables to the nonterminal procedures. This method also allows semantic actions to be implemented as PASCAL statements.

Using SYNGRAPH in interactive graphics programs involves a lexical specification, a grammar specification, and an underlying data abstraction which is the semantics of the interaction. The lexical specification defines the kinds of inputs that the user interface will use. The grammatical specification

describes the sequence of inputs required to formulate specific commands and the mapping of interactive commands to semantic actions. Semantic actions inserted in the grammar allow the user interface manager to invoke the application specific processing provided by the application procedures.

SYNGRAPH allows an interactive graphics application to be designed and implemented independently of the particular configuration of interactive devices that are available, it also allows the user interface to be separable from the application semantic routines. However the system can only be used by programmers, and to use it to the fullest extent, they have to be skilled in compiler construction, which is not true about most programmers. Thus, although SYNGRAPH allows the separation of the graphical user interface from the application, designing that interface remains the responsibility of the application programmer. Moreover, modifying or changing an existing user interface for an application is tedious, it has to involve the application programmer, and it always requires recompilation of the application programs.

UIMS (User interface Management System) [4] is a graphical user interface generator which is primarily concerned with producing menus. The system consists of two modules. The first, MENULAY, is a preprocessor which enables the user interface designer, using interactive graphics techniques, to design and specify the graphical layout and the functional relationships within and among the displays making up a menu based system. It is with this module that the application programmer establishes the relationships between what the user sees, does and hears, and the application-specific semantics underlying the program being implemented. Specifications made using this module (MENULAY) are converted into the C programming language and compiled through the use of a companion program, MAKEMENU. The resulting code can then be linked with application specific routines. Where the application designer specifies names of application-specific functions, the programs generated by MAKEMENU contain unresolved external references. By writing functions with the required names and referencing the appropriate file names when MAKEMENU is called, the application programmer can add any amount of application

specific programming to the layout, and sequence information specified by the designer.

The second module of UIMS is a run-time support package which handles interaction between the system and the user. It handles things such as event and hit detection, procedure invocation, and updating the display; all according to the schema specified using the preprocessor. The module also provides facilities for logging user interaction for later protocol analysis. This recording of sequential data about end-user interaction will be used to evaluate the interaction techniques used in an application program. UIMS supports two levels of users "novice" and "expert", but in both cases the users have to be application programmers, although at a later stage, a graphics artist can be involved to sketch the screens. The system only reflects one set of values at any one time, which makes the display of a multiple set of values, the responsibility of the application programmer. Hence, any user of UIMS has to have a programming background (except if the intention is only to sketch a menu after the user interface has been designed by the application

programmer), also a great deal of work is left for the application programmers to handle, if the application requires dealing with multiple sets of values and multiple windows. Another drawback of the system is that any change in the graphical user interface, except for changes in the sketching of a menu, will always require recompilation of the C language routines that comprise the graphical user interface, which implies that these will have to be linked again with the application programs.

Dialog [15] is a menu based user interface generator developed at Apollo computer Inc., which allows application programmers to design menu driven interfaces i.e. user interfaces which are based on choosing one of many options. The application programmer designs a menu by writing its description in a language which is similar to programming languages. Then the application programmer has to define and specify the relationship between the specified menus and the application in the same language after which both, the description of the menues and the specification of its relationship to the application have to be compiled by using the Dialog

Compiler, and linked to the applications object code. Dialog does not provide tools to design interactively the graphical parts of the menus in a graphical user intrface, rather it leaves that to the programmer who can use available graphics packages on the Apollo system to achieve that. Dialog suffers from many drawbacks. First, although the language used for specifying the user interface and its relationship to the application program is simple and of limited vocabulary, in order to use it, one must have a programming background. Second, the fact that the graphical part of the user interface is done using a different package, without allowing for interactive graphics techniques makes the system difficult to use in a grahical application, as the graphical parts of the user interface will actually have to be programmed by the application programmer, and hence any change in the graphical interface will have to involve the application programmer and the modification of the application. Third, even when modifying non graphical parts of the user interface, the interface description will have to be recompiled and then relinked to the application. In cases where the modifications are extensive, it might be necessary to recompile the whole

thing to incorporate the modifications in the user interface.

STAR[9], which was developed by XEROX on the XEROX STAR computer, a machine intended to perform applications related to office automation, is an example of a decent and a very succesfull user interface. The system utilizes a set of principles in its design, which can be applied in the design of any user interface. The design features of STAR are:

1)Use of a familiar user's conceptual model.
2)Seeing and pointing (everything is visible on the screen and can be moved by pointing).
3)What you see is what you get (the display screen portrays an accurate rendition of the printed page in an office environment).
4)Universal Commands (Commands that can be used throughout the system and perform the same way regardless of the type of object selected).
5)Consistency which asserts that mechanisms should be used in the same way whenever they occur, e.g the button which selects a character will also be used to select a

graphic line.

6)Simplicity ("simple things are simple, complex things are possible").

7)Modeless interaction: the interpretation of the keys does not depend on the mode or state the system is in.

8)User tailorability: STAR allows designers to tailor the appearance of the system according to user needs by providing many facilities for that.

The above implemented features make STAR one of the best user interfaces implemented, which probably explains the amount of time spent on its development, as STAR took about thirty man-years to be developed.

Reducing the amount of time required to develop a graphical user interface is one of the main goals of GUSIP, and GUSIP does that together with allowing for the seperation of the user interface from the application programs, and the ability to develop user interfaces without involving application programmers.

THE USER INTERFACE

A user interface can be defined as all the user and machine behavior that is observable by an external observer, and can be thought of as consisting of an input language for the user, not necessary a textual language, an output language for the machine, and a protocol for interaction.

GUSIP, by itself, is actually a component of a more general project that is intended to allow the generation of general user interfaces for computerized applications. The whole project consists of the following components:

-A graphical user interface generator.

-A menu based user interface generator.

-A Command language generator/interpreter.

-A help facility generator.

-A data validation and checking component.

The components are geared towards being as general as possible, and when combined together, they provide the user interface designer with powerful facilities to design a user interface of any application according to user needs and requirements. The user

interface can later be modified or changed altogether according to the changes in the user requirements, system operation or processing, or simply the level of sophistication of the users. One of the main project goals is to allow the generation of the user interface to the system itself. This is achived and demonstrated by the fact that the user interface to each component of the system is designed using the facilities of that component combined with those of the other components.

GUSIP is the component of the system which handles the generation and manipulation of the graphical aspects of the user interface. It allows the design and implementation of a graphical user interface of any application, starting from the drafting of the screens with graphical data values, and the screen sequence that is to be followed.

The command language generator and interpreter allows the user inte face designer to create a language to be used by users for interaction with the system. The commands can be generated by the system designer, then changed later according to end user's feedback. Simple

as well as complex commands can be generated by this component that allow performing almost any action required by the interface designer.

The menu based user interface generator allows for the design of user interfaces which do not require memorization. The user sees commands as menus and chooses the required action. Retrieval is also done using menus, from which the user selects the sought items of the stored data, in order to get more detailed information. Pop-up menus aid in the production of help facilities in the help facility generator.

The help facility generator is another very important component of a user interface generator, because the productivity of even the most advanced computerized system is diminished, to a great extent, if the user does not have the knowledge to use the available resources efficiently. Improving the ease of use of an application is the main goal of the help facility generator. This component allows an application to have help facilities which will tell the end user how to use the system, what are applicable inputs to a certain

query, how can one move to another part of the application...etc.

One component is the data validation component allowing the interface designer to check the input data before passing it to the application. It also allows adding or deleting certain constraints on the data which were not envisaged at the initial stages of the design, thus relieving the processing part of the application from the task of providing data validation routines which check for the input data against constraints that have been determined at the early stage of the design, and if any of those constraints is changed the data validation routines have to be modified, thus the application programmer has to be involved. The data validation component is used in the other components of the system which require user input.

The components of the whole user interface generator are intended to be independent for the time being, thus, the facilities provided by the other components could not be used for designing the user interface for GUSIP. Actually, those facilities which

were needed for making the user interface for GUSIP, were faked in such a way as to maintain the integrity and ease of use of the system (the main reason was the fact that the other components were not ready when GUSIP was implemented).

The few menus that GUSIP uses are programmed and not generated by the menu constructor component. They were included just to facilitate the use of GUSIP itself. Giving commands to GUSIP is done through using several keystrokes or mouse movements, as a command language was not generated for GUSIP using the command language generator, however, the way commands are implemented allows easy integration with a command language. Data validation was very limited in the design of GUSIP'S user interface, and in the graphical user interfaces which it generates. Also GUSIP itself, and hence the interfaces it generates did not have any help facilities as these are to be done by the Help Facility generator.

Terminology

Before describing the facilities that are provided by GUSIP and how they can be utilized. It is essential to clarify some of the main terms that are used in GUSIP itself or in explaining GUSIP'S capabilities and operation. The following explains some of the key terms according to the context in which they will be used:

Figure:

A figure is any graphical shape. It consists of one graphical element (a circle, or a rectangle), or of a collection of many graphical elements (many circles,rectangles, lines,...etc). Text which consists of letters, numbers, and punctuation is also considered as a figure and actually treated as such, since the system is mainly concerned with handling the graphical aspects of a user interface. Figures are usually constructed using the facilities of the graphics editor.

Predefined figure:

This is analogous to an encoded predefined value used in a textual context. A predefined figure is

constructed using the graphics editor and then reproduced any time after that whenever needed, by simply entering its number instead of drawing it altogether again (e.g in an application that allows user to construct houses, many predefined figures might refer to a circular window with crossbars, a third might refer to a sliding door, ..etc).

Interface Page:

This is a screen layout which represents a unit of the graphical user interface. It is designed by the user interface designer to show user input and application output in specific location and with certain constraints. It can be thought of as a graphical template which is used as a background for the input and output of the application's graphical data, in analogy to textual templates which are used as a background for the input and output of the application's textual data.

User Interface:

Within the context of GUSIP. This term will be used to refer to the set of all interface pages that comprise the graphical user interface for an application program. Where the general meaning of the term is intended, then it will be clear from the context, or otherwise clearly specified.

Input Location:

This is a position on an interface page that the interface designer specifies for the data entry and display of values for a specific variable that is used by the application program. This can be a fixed position or within certain limits. An interface screens can have as many input locations as desired by the interface designer (of course limited by the number of variables in a particular application).

Graphical Record:

This is the set of related graphical data values (of which some could be null), that can fill up all input locations on all the interface pages of

a user interface. From the application's side, one graphical record can be several logical or physical application records, or one or more graphical records can be combined to produce one logical or physical record.

Segment:

A segment is a physical manifestation of a figure, or a collection of figures. GUSIP stores graphical data as segments, which can be comprised of several segments.

GUSIP AND THE GRAPHICS DATA STRUCTURES

Introduction:

GUSIP aids in the design and implementation phases of a graphical user interface. In its current state, the system consists of two modules. The first is the main module, called the run time module, which handles all the run time aspects of the graphical user interface. It establishes the relationship between what the user sees and does, and the application specific semantics underlying the program being implemented.

The second module, called the graphics editor, enables the interface designer to sketch the layout of the graphical user interface by providing sophisticated interactive graphics facilities. It is with this module that the interface designer specifies the scenario of interaction between the end user and the application program.

Underlying both modules is the graphics package, which helps in the design of the data structure and in providing many of the graphics capabilities of GUSIP.

The Graphics Package:

The graphics package used by GUSIP is the DOMAIN 2D GRAPHICS METAFILE RESOURCE (hereafter referred to as GMR) which was developed by Apollo Computer Inc. for the Apollo computers. The package provides a versatile efficient tool for developing graphics application systems that store and display picture data. It is a collection of routines that provide the ability to create, display, edit, and store device-independent files of picture data. Following is a summary of the capabilities of the GMR package.

A)Storage Capabilities:

1) It offers a graphics system with memory which enables the integration of graphics data, textual data, display characteristics, editing, file storage, and hard copy output.

2)Virtual Storage of graphics files which

enables storing files of a very large size.

B)Modeling and Viewing Capabilities:

1)Commands which describe the least visible element of a picture.

2)Segmentation which enables grouping commands that make up seperate items of a picture, name the items and reuse them. Segmentation can be nested to any level.

3)Instancing which allows using a single sequence of commands several times and applying different attributes and transformations.

4)Picture modification by scaling to a larger or smaller size, or translation.

5)Multiple viewports which enable looking at more than one part of the picture simultaneously.

6)Commands which draw several predefined figures

like circles and lines.

7)Attributes which establish drawing characteristics such as line style and background before and after display.

C)Editing and Input/Output Capabilities:

These allow the easy creation of interactive graphics applications that change picture details interactively, and to use input devices such as a mouse or puck with easy interface, as well as transfer data to several hard-copy devices.

Data Structures:

The data structures used in GUSIP are affected by the GMR package, as the system uses many of the capabilities and characteristics of that package. The standard form of data storage is a metafile. A metafile is a device-independent collection of picture data (vector graphics and text) that can be displayed. The created metafiles are stored and then they are available

for redisplay, revision and reuse. They are not static copies of display bitmaps; rather, they contain lists of commands used to build a graphic image. Within a metafile commands are grouped into segments. Each segment named entity consisting of a sequence of commands. A segment can be referred to from another segment, in a manner analogous to a subroutine call. Individual commands within segments of a metafile descibe the least divisible components of a picture. There can be several types of commands:

-Primitive Commands describe a single least divisible, displayable component of a figure , e.g. a circle, a line, a rectangle...etc.

-Attribute commands contain values that specify the manner in which components of a figure are to be drawn, as the line style or text size.

-Instance commands cause references to be made to other segments. Instancing allows multiple uses of a single sequence of commands, with different transformations applied.

-Tag commands provide comments or other characteristics within a file or segment which do not affect the figures.

Every command is part of some segment. There are no commands outside of all segments.

As it will be seen later; the data structures that GUSIP imposes on its graphical user interfaces, attribute greatly to many of the desirable properties exhibited by those interfaces.

GUSIP: THE RUN TIME MODULE

The run time module of GUSIP handles and controls all interactions between the several components of an application's graphical user interface. It establishes the relationship between what the user sees and does and the application specific semantics underlying the system or program being implemented. Following is a description of the main activities that are performed by the run time module.

Screen Management:

Screen management is one of the basic and important activities that the run time module performs, particularly because GUSIP is intended to be used with interactive applications, where the screen is the main medium of interaction between the end user and the application. GUSIP gets all the parameters of a graphical user interface from the respective files, then it performs all the necessary screen management functions to display all the interface pages, each interface page is displayed in a seperate window, so as to provide the end user with the capability of inspecting any of the

graphical templates of an application program at any time during an interactive session. To do that the run time module creates the necessary windows to display all the interface graphical templates, then at the end of the interactive session, it deletes those windows. In the process of displaying the graphical templates, the run time module uses many of the routines that are provided by the GMR package. The run time module makes sure that the screen always contains the most up to date information, by reflecting all changes that are done to any figure directly on the screen, and by passing these changes immediately to all other templates that are affected.

Interaction between the End User and the Application Program:

When GUSIP is used with an application program(s) for the generation and management of a graphical user interface, it becomes the main communication link between the end user and that program. The run time module supports this by providing facilities for entering, modifying and displaying graphical records that are used

by a particular application. In each of these operations, the run time module passes necessary information to that application.

Entering data of a new graphical record involves starting with blank graphical templates, then the user will fill graphical (or textual) data values at input locations in these templates. Data values for a particular component of a template are usually entered in a particular location of the template which has been previously specified by the interface designer, these correspond to values of variables in the application. The run time module makes this correspondence by collecting the values of all components of the templates in the respective variables and passing them to the application program for processing. Displaying an existing graphical record is similar, although the source of data is different. The run time module receives values of the variables that comprise a graphical record for display, then it passes those as data values for the respective components of the templates, where the whole thing is displayed.

Modifying an existing graphical record involves the operations of display and data entry. First the run time module receives the value of the record to be modified, displays that record, then it allows the user to change data values for any desired template component(s), after which it has to collect the values again (after modification) in the respective variables and pass them back to the application program.

Throughout any of the above operations the run time module performs many screen management functions. These include displaying the templates of the interface (blank or filled with values), creating and deleting windows, and updating the screens properly according to the actions performed by the end user. Also it handles the interaction with the user by detecting actions and translating them to changes (additions, deletions, modifications, ..etc.) on the screen, thus giving the user a direct feedback about the status of the data that results from a particular event.

Interaction With the Graphics Package:

The GMR graphics package stores figures in graphics files as segments, where each segment can consist of other segments and/or graphics commands. The run time module accepts data from the end user of the application program, and handles the creation, retrieval and update of the segments in their respective graphics files. It also handles all data communication between the graphics package and the application program by passing the characteristics of the figures to the application programs in terms of meaningful parameters. GUSIP also translates user input commands, which are performed through menus, function keys or the mouse, to procedure calls in the graphics package which results in certain operations being performed on the graphics files or the displayed templates. The run time modules also manages the storage and retrieval of predefined figures when the user requests them for a particular location in a template. In this case the parameters of that figure are fetched from the application's graphics file, then the figure is displayed on the screen and its parametres are passed to the application program for processing.

To facilitate the processing of graphical data, the run time module provides the application program with a facility to inquire about the structure and display characteristics of any figure. In this case the figure is analyzed and its hierarchical structure and characteristics are passed to the application program. The structure includes the relative positions of segment to each other and any transformations that are applied to these segments. The display characteristics include the line style to be used when drawing the figure (dotted, dashed, solid,...etc), whether some elements are filled or not, and the location on which a specific figure is to be displayed. Besides the above, data on a graphics figure can include tag commands that are similar to comments in programs, these commands are also passed as part of the structure of a figure, to aid the application programmer in the classifications of certain figures and what they refer to, particularly because figures or segments can later be searched for using the values of the tag commands.

Drafting and Graphical Editing:

To facilitate the task of the application programmer and to allow easier interaction with graphical applications, GUSIP provides some elementary drafting and graphical editing capabilities for end users at run time. These capabilities allow the generation of new figures and the modification of existing ones. They include allowing the user to draw some elementary shapes, as circles, rectangles, ..., then using these shapes to draw more complex figures. They also provide scaling, rotation, translation and transformation operations to be applied to existing figures to modify them and change their appearance, characteristics and location on the display screen.

User Interface Management:

The run time module manages the user interface and makes it accessible to the end user for data entry, modification or retrieval. Depending on the way the graphical user interface was designed, entering data or modifying existing data is always started on the first

template. However, other templates are still accessible to the end user for inspection or modification, unless this accessibilty has been inhibited by the interface designer. Any restrictions or flow of control specified by the interface designer are taken into consideration and applied at run time, thus allowing the interface designer maximum control over the user interface.

Interfacing to components of the whole user interface:

The design of GUSIP, particularly the run time module takes into consideration any interfacing that needs to be done with the other components of the user interface. The run time module can interact with the data validation component of the user interface by passing to it data that is to be validated, then depending on the result; if the data is valid it is passed to the application program, otherwise, the run time module can communicate with the data validation module to take a certain action depending on the type of error that results from the invalid data.

GUSIP can accomodate the existance of help

facilities in the user interface, mainly because all displayed figures are stored on graphics files so they can be erased and redisplayed any time the need arises. Also because the graphics package allows drawing figures on a screen which are not related to something else that is displayed, besides the ability to manipulate windows as these can be used for processing help requests.

Menus can be easily incorporated in GUSIP's design, since the commands that it uses can always be picked by using menu structures. The menu construction component can thus translate the picked locations on the screen to command codes, or function key codes of the particular commands that are being displayed on the screen. Also the fact that the end user can pick any element on the screen and inquire about its characteristics facilitates tremendously the task of menu construction and handling.

The command language generator can make use of the powerful graphics handling capabilities in the design and interpretation of commands. Moreover, GUSIP lends itself to a command language since the commands can

be translated to GUSIP's command codes or function key codes as they are used and processed by the run time module.

GUSIP AND THE ENVIRONMENT OF THE GRAPHICAL USER INTERFACE

Introduction:

The environment in which any system exists and with which it interacts, affects to a large extent the behavior of that system. That's why GUSIP's relation with the environment of a graphical user interface helps to illustrate its generality, flexibility, user friendliness, and ease of use with virtually any application that requires the input, output, and processing of graphics data. It also explains why a huge amount of time can be saved in the design and implementation of a graphical user interface by using the facilities and tools provided by the system.

In general, the environment of a graphical user interface consists of the application program, the application programmer(s), and the end user. In addition to that, the internal data structure of the graphical user interface plays an important role in determining its efficiency, and can be the factor which enhances or limits the flexibility and generality of the user interface generator. The way in which GUSIP interacts

with its environment is thus an important element which determines its success or failure.

GUSIP and the application program:

Applications whose graphical user interfaces are implemented using GUSIP's facilities interact only with the run time module, though the interface itself is constructed by the graphics editor. In these applications GUSIP handles all interactions with the end user and performs all the communications necessary between the user and the application. Therefore, the application can concentrate on the processing aspects that are pertinent to its objectives. Data entry, modification and display are handled by GUSIP, and these are done according to the layout of the interface that was specified by the interface designer, the display always reflects the current status of the data and at the same time the application is always kept aware of any changes that occur to the data and any processing on that data that is requested to be done by the end user.

Hence, GUSIP acts as a communication tool between

the end user and the application. It allows making the design and implementation of the graphical user interface independent of the design and implementation of the application, except for some data specifications that are essential for proper running of the application and that should be taken into consideration by the interface designer while designing the user interface. Also the application program does not have to worry about where the data comes from and which devices were used for data entry, this is all handled by GUSIP which allows several input and output devices to be used in conjunction with a graphics applications, and hence the application can be device independent.

Thus, the application program can thus have several graphical user interfaces which can be utilized by a wide range of end user accoroding to their levels, experience, requirments and needs.

GUSIP and the Application Programmer:

The task of designing and implementing a suitable user interface for an application program has been

considered, until now, one of the trickiest and most cumbersome part of application development. This is due to the fact that user needs and requirments change depending on the particular users, even the same user might have different requirments and preferrence as he/she gains experience with a certain application. This has necessitated that application developers change the user interface of an application several times before a final acceptable version is obtained. This was often not an easy task to do since the user interface was scattered throughout the application, and any changes that need to be done to the user interface, might involve major changes in the application program(s).

GUSIP relieves the application programmer of graphics applications from the tedious task of designing (and redesigning or modifying) the user interface for their applications and incorporating that in the applications by providing the necessary tools and facilities for the generation and implementation of those user interfaces. The application programmer can design and develop the application independently of the user interface and safely assume that the required data will

be available at the time it is needed, without having to take into consideration the means of receiving input from the users and relaying output to them, or even which devices will be used for input and output to that application. Also, the graphics application program does not have to take into account the graphical characteristics of data or the structure under which this data is being stored and/or displayed as all of this is handled by GUSIP.

The graphical user interface is drafted and drawn using the graphics editor which provides the interface designer with very powerful tools for the design of almost any graphical layout. This is associated with the application without the need to involve the application programmer. This interface can be modified or replaced in the future, without the involvement of the application programmer as this will not have any implication on the program's design, programming or performance.

As for the structure under which data is stored GUSIP uses the facilities provided by the GMR graphics package to store the graphics data in files with

hierarchic segment structure, then it uses routines available in GMR to retrieve, display, or modify these files. By utilizing the GMR graphics package which stores data in a metafile that is device independent, the application programmer can develop the application without worrying about the devices used for interaction with users.

The application programmer includes only the run time module in the application as an available library of programs and/or procedures, which handle all aspects of the graphical user interface, and can use some of these procedures in the application to perform certain required tasks, or to do some processing on the graphics data. When the application programmer integrates GUSIP in the application, the user interface looks as if it were part of the application itself, and at the same time this application can be run using many different graphical user interfaces without the application programmer having to change a single statement in the programs.

GUSIP and The End User:

Users of computer application have always wanted to interact with those applications in the way that suits them best, and wished that the user interfaces used for the interaction are tailored according to their needs and requirements. Also, many times a user interface requires modification whenever users preferences, needs, requirements or environment undergo any change. These requirements by users have been taken into consideration by many system developers, as user acceptability of any system depends on how suitable it is to user requirements. However, not many have succeded in satisfying these requirements mostly because the current understanding of human-computer interaction is very limited. One cannot sit down and design an interface and know, apriori, how well it will perform. Therefore, one must accept at the outset the inevitable intertwining of specification, design and implementation. Design then becomes an iterative process, each iteration consisting of three phases: design, implementation and evaluation. The motivation to develop improved tools can therefore be seen as a desire to increase the number of iteratives that

one can afford to pass through in this loop.

GUSIP is one system which provides these tools, and a consequence of using it is an improvement in the quality of the user interface that is produced. To the end user the presence of GUSIP is transparent, and the impression is that the graphical interface is part of the application program. Any modification to that interface that is introduced at a later stage makes the users think that the whole application has been modified to suit their needs. This enhances their acceptability of the application and their willingness to use it. Also by introducing this flexibility in the design and implementation of the user interface of a graphics application, the number of times that one can afford to pass through in the above mentioned loop of specification design and implementation is increased, this makes users feel at ease knowing that no interface has to be lived with for life and that the user interface can accomodate any changes that might be required in the future and these changes can be incorporated without the prohibitive and frustrating factors of difficult and elaborate procedures, too much time and high cost.

The user interface designer acts as an end user for GUSIP itself, and by using the system can have greater flexibility in the design and redesign process of the user interface for graphical applications. This is mainly due to two factors: one is the availability of very powerful graphical tools for generating the graphical interface, and two because the designer can produce one interface, use it for a while then modify it according to user's feedback then repeat the whole process until a final version is produced which suits most (if not all) users. Besides, that any changes in user requirements that might arise in the future can be taken into consideration, and incorporated in the interface easily, without the need to involve the application programmer, and with no changes that need to be done to the application, as the interface and the application are very loosely coupled using GUSIP, whereas they used to be very closely coupled in the past. Actually, GUSIP'S ease of use and extremely understandable concepts enable many users to utilize it, with some training, for designing user interfaces for graphics applications that they want to use according to

their preferences, convenience, and taste.

GUSIP and the Internal Structure of Data:

The internal structure of data that is exhibited by a user interface can enhance or limit the flexibility and power of any system. This has affected the design of the data structure that is exhibited by user interfaces which are generated by GUSIP. For each application's graphics data, a graphics file is maintained which has a dynamic structure and size that allows it to store all the graphics data for that particular application. This file contains all the predefined figures as well as all the figures and shapes that are constructed by the end users of the application. It is randomly accessed by GUSIP, hence serving as a graphics data dictionary for the application besides being the main graphics data file.

Each user interface that is generated for an application has associated with it a graphics file which contains all the figures and shapes that comprise the layout of the graphical user interface. Thus each

application will have associated with it as many such files as there are user interfaces that have been generated for it. This structure allows the graphical user interface to be accessed and manipulated independently of the data and the application, in addition to that, any modifications or changes that are applied to it do not affect neither the application's data nor the application's programs.

Having one main graphics file for each application and several ones for the user interfaces enables GUSIP to utilize several user interfaces at the same time, since the data exists independently of the user interface layout, and displaying it is handled properly according to the graphical user interface that is being used at that time. This does not mean however that that there is any limit on the number of files that an application can have, but rather is meant for efficiency purposes in terms of storage and processing of graphics data. Also, this enables the application that uses other files which are intended to store references to the graphics data (or the data itself) to store in those files only the segment numbers of the figures under

consideration. These will then be more compact, and can be treated as regular textual data files since all the graphical properties of the data are handled by GUSIP in the respective files.

This structure that is imposed on the graphical user interface enhances tremendously the generality and ease of use that are attributed to the interfaces generated by GUSIP, also it helps to relieve the programmers from worrying about the complex issues of handling and manipulating graphical data.

Thus GUSIP'S interfaces interact directly, easily, and successfully with all the components of their environment to produce to end users a quality user interface that fits perfectly their needs and requirements.

GUSIP: THE GRAPHICS EDITOR

One of the most important goals of GUSIP is to allow the end user to have control over the user inreface; by providing that user with powerful and comfortable facilities that do no require sophisticated computer knowledge to manipulate. This is achieved by the facilities provided by the graphics editor component of the system which is used for the design of the interface layout, the specification of control tasks, and the design of predefined figures. The output of the editor is a set of files which contain a description of all the information necessary to define the display, these files are used by the run time module while an application is being run. A description of the main functions performed by the graphics editor explains the flexibility and power that it gives to the designer of a graphical user interace.

Construction of The Interface Layout:

The graphics editor allows the interface designer to specify the geometry of the graphical user interface by allowing unlimited access to all parts of the screen,

and hence the interface designer has full screen graphics editing capabilities under his/her disposal, to perform any geometrical operation necessary on the pages of the user interface. The pages of the user interface are constructed one by one, however, at any time all these pages are available for review and inspection by the designer. This is easily done by switching between the windows in which the interface pages are displayed. The designer need not know any programming, and basic knowledge of how to use input/output devices is all that is needed for this construction to be done. Besides having very powerful graphical construction tools to specify the layout of the interface, the interface designer can use graphical editing facilities of the graphics editor to modify, alter, or redesign any number of interface pages. Elements of geometry that are specified at the construction stage are the location of each variable on the particular screen displaying the variable's interface page, the restrictions and/or limitations that are to be applied to values of that variable (as length, width, specific shape,...) and the relationship, if any, that values for this variable might have with values of other variables.

Specification of Control Tasks:

There are other aspects of generating a user interface than specifying the geometry, these are interface control tasks and are also handled by the graphics editor. A very important task of a user interface generator is to associate a user interface with an application. Although the actual association is done at run time, the parameters of this association have to be specified at the design stage of the graphical interface. The interface designer determines those parameters and relays them to the editor. There can be several graphical interfaces to an application, but at any one time only one interface has to be associated to be chosen by default when that application is executed, the editor maintains this information in a specific file where it records all the available interfaces to an application, and the file name of the interface that is to be used by default with that application. This association can be changed at any time by the interface designer; either by modifying the graphical interface associated to the application, or by associating with

that application a different user interface altogether.

Another parameter of associating a user interface with an application is the correspondence between the values that will be input (output) at specified locations on each interface page and the variables used within the application. This correspondence is established, in GUSIP through the use of an array that holds the segment numbers of graphical figures and each element of the array corresponds to a variable of the application. Thus, the correspondence is determined by specifying at each input (output) location the subscript of the variable which will hold the value displayed at that location.

These facilities, provided by the graphics editor, allow the application programmer to write the application programs assuming certain information in a certain format from the graphical user interface. Then the interface designer (or end user) can modify the interface layout, the geometry of the screens, or actually exchange two user interfaces, and as long as the user does not modify any aspect of the user interface

that affects the semantics of the underlying application, then that application program need not be changed. The format in which the information is communicated to the application is under the control of GUSIP, not the user. Thus for any graphics application, the interface designer (or end user) has control over the user interface without modification of the underlying program.

Construction of Predefined Figures:

The construction or drawing of predefined figures requires the same graphical drafting and editing capabilities that are needed when specifying the layout of the graphical user interface, hence this task was associated with the graphics editor to make use of its powerful capabilities.

A predefined figure is treated by the graphics editor similar to the layout of an interface page. Thus, the end users or the interface designers can use the capabilities of specifying the layout of an interface page to draft and edit their predefined figures. However, there is one difference between predefined

figures and interface pages, namely that an interface page's graphical layout is stored in the graphics file of the interface, whereas the graphical layout of a predefined figure is stored in the graphics file of the application itself. This is a very important and essential aspect of handling predefined figures because it allows predefined figures to be used within the application, no matter what user interface is associated with the application at that time.

Graphical Capabilities:

The graphical handling capabilities that were embeded in the graphics editor allow the user full control over the layout of the graphical interface or the drawing of predefined figures. By using the mouse and some other keyboard keys GUSIP's user can draw almost any figure, and by combining some predefined shapes, (as circles, rectangles, arcs, lines, polygons,...), spline curve fitting, text or freeform drawing with the ability to graphically edit any drawn part at any time by using the graphical editing facilities of the editor, namely moving and copying figures and/or shapes, applying

certain transformations on any thing on the screen (translation, rotation or scaling) and the ability to erase any part of the screen to redraw it altogether.

FEATURES AND SIGNIFICANCE

Portability:

It is important that a uer interface generator be portable in a number of senses. GUSIP satisfies several portability criteria in theory and practice.

Input Device independence:

The user interface generator must be able to support alternative input techniques. The primary input devices used in GUSIP are the mouse and the terminal's keyboard. Other devices are now available and can be supported by the GMR package, a graphics tablet can be used in place of the mouse, and actually any input device, that can be supported by the graphics package,can be used with the system. The run time module, however, can be driven by virtually any event generating device that is supported by GMR.

Output Device Independence:

The user interface generator must also be able to support a number of different output devices with different characteristics. In the case of GUSIP the device independence is achieved by using the GMR graphics package which can support varied output devices as a high resolution monochrome display, a high resolution color display and several types of laser printers.

Language Independence:

As a further level of portability, the user interface generator should be structured to allow programs written in many programming languages to use it. Using the capabilities of the APOLLO Domain operating system programs written in C or FORTRAN can still use GUSIP to generate their graphical user interfaces. The way to include the needed routines, however, should be modified slightly, so that it is first

compiled then the object code is linked to the object code of the application.

Machine Independence:

The system, as it is now, can not be transported to other than APOLLO machines. Meanwhile, the graphics package, the langauge compilers and the operating system on the APOLLO machines are written in PASCAL. This might help, later on, in providing a means for making the system, together with the GMR package, run on several machines.

A Recursive System:

A very important feature of GUSIP is that it treats itself as an application program and hence its graphical user interface can be designed using its own facilities. This provides the system with tremendous power because of the recursiveness involved, since any specified features can be used in a later stage to develop

more complex and elaborate features, and because of this recursiveness the system's facilities and capabilities can be improved upon almost indefinitely.

Comparison with Other Systems:

GUSIP is not the only and probably not the best graphical user interface generator available, however, a comparison with other existing systems might emphasize its distinctive advantages. The most distinguishing feature of GUSIP is its natural way of integrating the designed graphical specifications with the application programs. By way of comparison, three systems will be reviewed: SYNGRAPH [8, 9], UIMS [4], and DIALOG [15].

SYNGRAPH:

SYNGRAPH uses PASCAL procedure definitions for the utilization of interactive commands in the application program. GUSIP, is a significant improvement over this idea, in that

the user interface and the interaction relationships are specified in the very way in which the end user will interact with the application, i.e by using the mouse and the keyboard. Also, SYNGRAPH is meant to be used only by application programmers, whereas GUSIP does not have that restriction and can actually be used by the end users themselves, with little training.

UIMS:

UIMS also suffers from the fact that the application programmers have to be involved, at least at one stage, in the design of the user interface. Besides, the system cannot deal with multiple values or multiple windows and any change in the user interface usually requires a recompilation of the application program. GUSIP overcomes these drawbacks by allowing end users to design their own user interfaces, providing a window management component for the generation and handling of windows, and since the interface

is stored in a non-program file then it does not require any recompilation of the application program if it undergoes any changes.

DIALOG:

DIALOG adopts a common approach for the specification of a user interface, namely it uses a formal language for the specification of the geometry and logic of menu based user interfaces. This approach is very limited for the geometric specifications, as the user must learn the language, which involves learning keywords and syntax that really contribute nothing to the user interface [3]. This also necessates that the application be recompiled any time the user interface is modified, no matter how small or insignificant that modification was. From the mentioned characteristics of GUSIP, it is evident how it overcomes these drawbacks of DIALOG, namely by allowing the specification of the user interface to be done using the same interaction techniques, and by making the interface

independent of the application program and the application programmer.

CONCLUSION, LIMITATIONS AND FUTURE CONSIDERATIONS

Using GUSIP allows application programmers to design and write their application programs without worrying about the user interface aspects of those applications. Instead they can concentrate on the processing part of the application, plus some interfacing with GUSIP. GUSIP provides both the application programmer and the user interface designer with many facilities specially graphics handling capabilities, but it should be kept in mind that GUSIP is only one component of a group of projects that are intended to manage generalized user interfaces, and as such will never be complete until all parts have been implemented and integrated into one product. The user interface to each of the components is built by using the facilities of the other components as well as its own facilities. The above explains why GUSIP's interface is not as fancy as expected, and why GUSIP lacks some of the desirable features of interactive graphics and help facilities like menus.

However GUSIP is not a perfect system and actually has many limitations which if resolved can help

in enhancing the tools that will be available for application programmers and interface designers for the development and management of flexible user friendly, and sophisticated user interfaces. These limitations include:

a)Machine Dependence:

Although GUSIP can use several input and output devices, it is bound to be used only on the Apollo machines. This limits its usage by a wide range of users and applications that might require the system to be used on other machines.

b)Colored Graphics:

The GMR graphics package supports colored graphics, but this facility was not use in the current implementation of GUSIP which cannot support applications that require handling of colored graphics, a desirable feature to have in modern graphical applications.

c) Two Dimensional Graphics:

Using the recursive feature of the system, facilities can be developed to handle and manipulate successfully three dimensional graphics, but the present version supports only two dimensional graphics.

d)Graphics Package Dependence:

GUSIP is dependent on the GMR graphics package which is not the industry standard in computer graphics, and a very important enhancement to its usage and portability can be attained by resolving this dependence and allowing the system to utilize other available graphics packages; like CORE and GKS.

The above issues provide a basis for work on improvements on the system or for general experimentation and research in this field.

APPENDIX

USER MANUAL FOR GUSIP

There are several types of tentative users to GUSIP and each user needs to use a specific part of the system, thus requiring a different user manual. The application programmer who wants the graphical user interface for a program to be handled by GUSIP should be able to communicate with the run time module of GUSIP. A user interface designer needs to be able to use the graphics editor and relate input locations with variables in the application itself. End users need to be able to use the facilities of GUSIP to enter data to certain applications.

Following is a brief description of how different types of users can achieve their aims of using GUSIP's facilities. This is preceded by an explanation of the terminology used.

SYMBOL or TERM	MEANING
'A'	Capital letter 'A'
'a'	Small letter 'a'
a	Capital or small 'a'

The above three are the same for any letter.

Fn	Function key number n (e.g F1)
sFn	Shifted function key n
[Action]	Predefined function key on keyboard (e.g. [help])
M1	Left button on mouse
M2	Middle button on mouse
M3	Right button on mouse
Locator	The type of input which occurs when the mouse is moved

GUSIP For the Application Programmer:

a) Insert File:

The application programmer should include the file (GUSIP.ins.runtime) in the application which is intended to use GUSIP for implementing its interface. This is done as follows:

```
% Include "GUSIP.INS.RUNTIME";
```

b) Passing values:

Passing values between GUSIP and the application is done by an array of Components, each component is a PASCAL record structure which contains two fields.

1) Type of input: integer which specifies the type of the input or output data as follows:

0 : no input

1 : predefined figure

2 : text

3 : user drawn figure

This is defined as:

TYPE-OF-INPUT:0..3;

2) Segment number: the segment identification number of the segment which was constructed as a result of a user input or the display of the record. This field is of type GM-$SEGMENT-ID-T. IN case the input type is zero then the segment number is undefined. This field is defined as:

SEG-NO: GM-$SEGMENT-ID-T;

c) Procedures Needed:

The application program basically needs two procedures to interact with GUSIP. The first gets input values for a graphics record, besides being used to display graphical records. The

procedure call is:

```
GUSIP-$SEGMENTS(operation, N, recordsegarray);
```

Where "operation" is the operation that the application intends to be performed on the records passed to GUSIP. This is coded as follows:

1: Input values for the record.

2: Display the record with possible modification.

3: Display the record.

N is the number of figures being passed and "recordsegarray"is an array of graphical components as described above.

The second GUSIP procedure, needed by the application, inquires about the structure of a segment. The procedure call is:

```
GUSIP-$INQ-SEG(SEG-ID, figpointer);
```

Where "SEG-ID" is the identification number of the segment to be examined; this is of type GM-$SEGMENT-ID-T, and "figpointer" is a pointer

to the first element of a linked list of elements that represents the structure of the figure, and each element of that list represents the drawing of a graphical element. "Figpointer" is of type GUSIP-$POINTER.

GUSIP for The Interface Designer:

The interface designer mainly uses the graphics editor of GUSIP, the basic operations used are:

a) Drawing figures:

The following are the basic drawing facilities and a description of how they can be used.

1. Draw a line: Press F1, locate first point on the screen and press M1, then locate second point on the screen and press M1.

2. Draw a line parallel to X-axis: press X then proceed as in 1 above.

3. Draw a line parallel to Y-axis: press y and proceed as in 1 above.

4. Draw a rectangle: press F2 then locate the positions of two diagonally opposite corners by pressing M1 after locating each point.

5. Draw a circle: if the center and a point on the circumference are known then press F3 and locate these two points by pressing M1 as above, otherwise if it is desired to fit the circle between two points; press F4 then proceed in the same way to locate the two points.

6. Draw an arc: Press F5 then locate three points on the circumference of the arc by pressing M1 after locating each point.

7. Fit a spline through a set of points: press F6 and locate the set of points to be fit by pressing M1 after each and pressing M2 when the set of points is exhausted.

8. Draw a polygon: Press p then locate the set of vertices of the polygon by pressing M1 after each point and pressing M2 when the set of points is exhausted.

9. Draw a free form: Press F8 then press M1 to start drawing, move the locator around the screen as desired, when finished press M1 again.

10. Press F7 then locate where to enter the text by pressing M1, enter the text then press M1 again when the text to be entered is exhausted.

In addition to the above, to start constructing a figure which might consists of many shapes, press 'f' and at the end of drawing that figure press 'sF1'.

Editing Figures:

To edit a figure press 'e' when the cursor is at

that figure. The figure will be outlined for a short time, after that all drawing or editing operations will be considered to affect shapes only in that figure. If no figure is selected with edit operations (i.e. no figure is being constructed or edited) then edit operations apply to complete figures.

The edit operations are:

1. Move: Press 'm' with cursor at the shape to be moved, the shape is outlined for a short time, move the locator until the shape is at the desired position on screen then press M1.

2. Copy: Press 'c' with the cursor at the shape to be copied, it will be outlined for a short time, move the locator until the shape is at the desired location and press M1.

3. Scale: Pres 's' with the cursor at the shape to be scaled, it will be outlined for a short time, move the locator to the right to grow the shape and to the left to shrink until it reaches the required size, then press M1.

4. Rotate: press 'r' with the cursor at the shape to be rotated, it will be outlined for a short period, then move the locator to rotate it until the desired orientation is achieved, then press M1.

5. Delete: press 'd' with the cursor at the shape to be deleted, it will be outlined for a short period, press M1 and the shape will be deleted. When the screen is crowded with figures; it is advisable to move the shape or figure to be deleted, so as to isolate it from other and avoid any mix up.

Input Location:

To determine input locations and associated variables; position the cursor at the location where data is intended to be entered and press 'i', then enter the subscript of the variable in which the value entered at this input location will be stored.

GUSIP and The End User:

The end user only interacts with the run time module of GUSIP when using an application, and this offers the user basic graphics drawing and editing facilities as mentioned earlier. This interaction is done through the underlying application, although the semantics of the interaction are determined by the interface designer.

REFERENCES :

1. L. J. Bass and R. E. Bunker "A Generalized User Interface for Application Programs". CACM vol. 24 no. 12 pp 796-800, Dec 1981.

2. L. J. Bass "A Generalized User Interface for Application Programs (II)". CACM vol 28 no. 6 pp 617-627, June 1985.

3. L. J. Bass "An Approach to User Specification of Interactive Display Systems". IEEE Transactions on Software Engineering vol. 11 no. 8 pp 686-698, Aug 1985.

4. W. Buxton, M. Lamb, D. Sherman and K. C. Smith "Towards a Comprehensive User Interface Management System". Computer Graphics vol. 17 no. 3 pp 35-42, July 1983.

5. Uli Chi "Formal Specifications of User Interfaces: A Comparison and Evaluation of Four Axiomatic Approaches". IEEE Transactions on Software Engineering vol. 11 no. 8 pp 671-685, Aug 1985.

6. P. Feiler, G. Kaiser "Display Oriented Structure Manipulation in a Multipurpose System". Proc. IEEE COMPSAC 83, pp 40-48.

7. J. Foley and A. Van Dam "Fundamentals Of Interactive Computer Graphics". Chap. 6 pp 217-243. Addison Wesley Publishing Co. 1982.

8. D. Olsen Jr., E. Dempsey "Syntax Directed Graphical Interaction". Proc. Symp. Prog. Lang. Issues, ACM San Francisco CA pp 112-117, June 1983.

9. D. Olsen Jr., E. Dempsey "SYNGRAPH: A Graphical User Interface Generator". Computer Graphics vol. 17 no. 3 pp 43-50, July 1983.

10. D. C. Smith, C. Irby, R. Kimball and B. Verplank "Designing the STAR User Interface". BYTE vol. 7 no. 4 pp 242-282, Apr. 1982.

11. H. Strubbe "Kernel for a Responsive and Graphical User Interface". Software Practice and Experience vol. 13 pp 1033-1042, 1983.

12. T. Takala "User Interface Management System With Geometric Modeling Capability: A CAD System's Framework". Proc. Computer Graphics, Tokyo 1984.

13. Programming with DOMAIN 2D Graphics Metafile Resource. Apollo Computer Inc. Release 9.0, July 1985.

14. DOMAIN System Call Reference, vol. 1 & 2. Apollo Computer Inc. Release 9.0, July 1985.

15. DOMAIN DIALOG User's Guide. Apollo Computer Inc. Revision 0.0 July 1985.