A GRAPHICAL INTERFACE FOR THE INTEGRATION OF ALGORITHM ANIMATIONS

A Thesis

by

CHRISTOPHER JAMES RODA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 1992

Major Subject: Computer Science

A GRAPHICAL INTERFACE FOR THE

INTEGRATION OF ALGORITHM ANIMATIONS

A Thesis

by

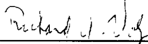
CHRISTOPHER JAMES RODA

Approved as to style and content by:

James Abello
(Chair of Committee)

Paul Nelson
(Member)

Thomas Linehan
(Member)



Richard Volz
(Head of Department)

December 1992

ABSTRACT

A Graphical Interface for the

Integration of Algorithm Animations. (December 1992)

Christopher James Roda. B.S., The Ohio State University

Chair of Advisory Committee: Dr. James Abello

Recently, the computer science community has seen the emergence of several algorithm animation systems created to help in the understanding of algorithmic principles and techniques. Examples of such systems are AGE, Balsa and Tango. With time, algorithms have become more and more complex. They tend to be quite interelated and their implementations may be distributed over multiple machines. Under these circumstances, a need has arisen for an easy to use interface that allows the user to express and control different levels of algorithmic interactions.

This thesis proposes a graphical interface that facilitates the integration of previously defined algorithm animations and other animation independent applications. Since modularity has been regarded as an essential design principle, each of the basic objects the interface manages has been categorized associated with a set of well defined operations.

From the user's point of view, our graphical interface behaves as a graph editor whose vertices correspond to previously defined applications and the edges correspond to the flow and progression of data among them.

From the internal perspective, five entities compose a three layered communication.

The first layer consists of a User Event Handler, an Object Table, and several Interface Communication mechanisms. These entities form a triangle comprising the foundation of the interface. The middle layer consists of several Communication Agents which receive instructions from the foundation, manipulate child application processes and transmit data amongst each other. Finally, several I/O Masters converse with application processes, Communication Agents and the foundation and act as the links between different animation environments and the interface.

A prototype for this interface has been created on UNIX with the X Window environment using the OSF/Motif toolkit. The source code was developed using the C++ language, and Sun Sparc Stations were used.

One of the main contributions of this work is the interface methodology. It enables user events to interactively manipulate and monitor the communication among previously defined processes. This is achieved with the assistance of a specialized database and a process communication tool.

We believe this effort provides a useful set of principles which can be used to guide the design of interfaces whose main function is to provide a link between different algorithm animation systems and other applications.

ACKNOWLEDGMENTS

This thesis is the result of many long hours of work spent by many people and I would like to thank all those who helped and those who I have failed to mention. Thanks! I couldn't have done it without your help.

I would especially like to thank Dr. James Abello for his encouragement and many late night hours spent for this effort. Likewise, I would like to thank my other committee members Dr. Paul Nelson and Dr. Thomas Linehan for their consistent patience and support.

Special thanks goes to the UNIX gurus Craig Smith, Jeff Waller and Ron Theriault whose expertise helped me through the most difficult Unix puzzles. Likewise, appreciation goes to Andy Dennis for insight on the delicate nature of bitmaps. A special thanks goes to Tim Veatch for his help with AGE and all the AGE client creators: Lucero Torres, Don Sonom, Craig Smith, Beata Bloch, and Matt Kernek, for their valuable contributions.

I also thank Dr. Abello, Craig Smith and Shawn Carlow whose reviews of preliminary versions of this document greatly improved its content and readability.

Finally, I am very grateful to my parents and the gang, Greg Schmidt, Andy Dennis and Shawn Carlow for their seemingly endless moral support and especially Ha Nguyen who provided the strength, love and light to carry me through my darkest hours.

TABLE OF CONTENTS

Page

ABSTRACT iii

ACKNOWLEDGMENTS v

TABLE OF CONTENTS vi

LIST OF FIGURES ix

LIST OF TABLES xi

CHAPTER

I INTRODUCTION 1

I.A Motivation 1

I.B Approach 2

I.C Implementation 6

I.D Related Work 7

I.E Thesis Outline 9

II INTERFACE EXTERNAL VIEW 10

II.A Interface Access 11

II.B Sample Session 11

II.C Administrative View 16

III INTERNAL DESCRIPTION 20

TABLE OF CONTENTS (CONTINUED)

CHAPTER PAGE

III.A Global Interface View 20

III.B Levels of Interface Communication 22

III.C The First Level of Interface Communication 24

III.D The Second Level of Interface Communication 34

III.E The Third Level of Interface Communication 41

IV CONCLUSION 44

IV.A Results 44

IV.B Future Enhancements 49

REFERENCES 53

APPENDIX

A USER MANUAL 54

A.A Interface Access 54

A.B External View 54

A.C Menu Bar 56

A.D Animation Display Area 57

A.E Control 57

A.F Canvas 68

A.G Error Box 72

B APPLICATION REGISTRATION 73

TABLE OF CONTENTS (CONTINUED)

CHAPTER		PAGE
	B.A Needed Information	73
	B.B Catalogs	74
	B.C Header Files	78
	B.D Application Registration	81
	B.E Application Input	82
	B.F Application Output	83
	B.G Application Self Loops	84
C	I/O MASTER CREATION	85
	C.A Command Router	85
	C.B Environment Liaison	86
	C.C Output Dispatch	87
D	GLOSSARY	89
VITA		91

LIST OF FIGURES

FIGURE		Page
1	A Simple Animation Digraph	4
2	A More Sophisticated Animation Digraph	5
3	Interface Appearance Upon Invocation	10
4	Interface with AGE Animation Digraph	12
5	A Sample AGE Animation Digraph	14
6	Catalog Loading Menu	18
7	Query Box	19
8	Global Interface View	20
9	AGE Interface View	21
10	Interface With AGE Animation Digraph	22
11	The Three Levels of Interface Communication	23
12	The First Level of Interface Communication	25
13	Internal Interface Message Flow	27
14	Interface Status Manager	29
15	Object Table Entry Information Flow	30
16	The Second Level of Interface Communication	34
17	UCA Bulletin Board	37
18	Multiple Process Interaction	38
19	File Input and File Output	40
20	The Third Level of Interface Communication	41

LIST OF FIGURES (CONTINUED)

FIGURE PAGE

21 Interface Appearance Upon Invocation 55

22 Menu Bar 56

23 Unix Access Menu 56

24 Interface Information Menu 57

25 The Control Subwindow 58

26 The Agents Submenu 59

27 The Sysload Popup Window 60

28 Query Box 61

29 File I/O Submenu 63

30 File Load Window 64

31 Interface Status Manager 67

32 Animation Digraph Icon Menu 71

33 Catalog Loading Menu 77

34 Sample Graph 79

35 Query Box 82

LIST OF TABLES

TABLE		Page
I	Icon Color Status	69
II	Format String Decomposition	78
III	Major Functions of CommandInterpreter.c	86
IV	Major Functions of AGELiaison.c	87
V	Major Functions of OutputDispatch.c	88

CHAPTER I

INTRODUCTION

I.A Motivation

Recently, the computer science community has seen the emergence of several algorithm animation systems created to help in the understanding of algorithmic principles and techniques. Systems such as *AGE* created by Abello, Sudarsky, Veatch and Waller [1], *Tango* by Stasko[10] and *Balsa II* by Brown [3] have contributed significantly not only to the teaching environment but also to the research community. As these systems have become more available, a large number of algorithm animations have appeared. With time, algorithms have become more and more complex and they tend to be extremely interrelated and possibly distributed over multiple machines. With this in mind, *a powerful but simple to use graphical interface is needed to express and control different levels of algorithmic interactions.*

To understand the duties of such an interface, it is important to ask the following questions:

1. How can this interface be designed to maximize ease of use?
2. What can the interface provide for fast creation of algorithm animations?
3. If there is to be communication of data, where is the data coming from and where is it going to?

Answers to the previous questions define the desired interface behavior and this in turn

Journal model is *IEEE Transactions on Computers*

Answers to the previous questions define the desired interface behavior and this in turn determines the objects that such an interface must manipulate.

The design of the interface reflects these duties in its implementation. A novice user is able to operate the interface and its available tools with little or no training. To achieve this, the interface is self defined with informational labels, help windows and easy to follow instructions to help the user integrate previously defined algorithm animations. While the interface is structured to guide the user along each step of an interactive session, skilled users are not hindered by the aids.

The effectiveness of the interface is judged on how "quickly" it operates. This is a challenging factor since the interface would most probably interact with a local network and the operational time is thus controlled by the network environment. In consideration of this fact, the interface needs to be designed to minimize the waiting time for any network operation. Operations that force the users to wait more than a few seconds diminish the effectiveness of the interface.

I.B Approach

Our method, to provide some answers to the challenge posed by the task of designing a user friendly interface to integrate algorithm animations, is to classify the basic objects that have to be dealt with. Each object has a set of well defined operations associated with it. The objects are categorized into algorithm animation environments, algorithm animations created to execute under those environments (clients), and other environment independent applications. Examples of the latter include applications needed for input, output and visual display. Communicated data may come from a keyboard, an input file, a

visual display device, an algorithm client or other application. The destination of the data may be to output files, other animation clients, applications, visual displays or to hard copy devices.

Our approach is to represent a typical interface input as a directed graph, called hereafter an *interface animation digraph*, or simply an animation digraph [1]. An animation digraph is composed of vertices and edges. Each vertex has an (animated) iconic visual representation and has an associated set of specified internal processes. The edges connecting the icons correspond to directional flows of data from one process to another and represent the schematic order of sequential execution. The arrows on the edges indicate the direction in which data flows along the paths.

The main task of the interface consists of interpreting an input animation digraph, activating the processes associated with each icon, and providing the necessary structures to guarantee the assigned task is to be completed successfully or to be aborted gracefully in case of abnormal conditions. Upon execution, the interface parses the animation digraph, identifies its associated objects, prepares a *communication digraph* for those objects, invokes the corresponding processes and places them in an execution waiting state.

The interface behaves both in a batch and on-line manner. The user, with the help of the graphical interface, is able to compose an animation digraph using previously defined objects. The interface creates an interprocess communication mechanism in such a way that an animation digraph can be added to, subtracted from, or its execution can be started and stopped at any particular time. In summary, the user has the ability to interact with an animation digraph, controlling its execution, termination and configuration.

The user may select any icon as the initial vertex within the animation digraph. Ex-

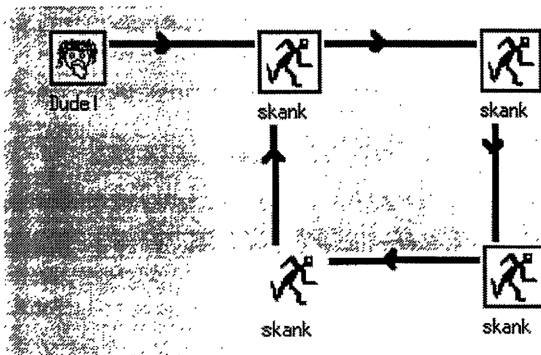


Fig. 1. A Simple Animation Digraph

ecution starts at some internally selected icon's process. Once this process terminates or transmits data, control is transferred to the set of selected processes associated with the neighboring icons. If data is passed, the information is delivered to the set of selected icons. Execution continues as long as there is a control or data path to be followed. The user may, however, terminate execution at any time. When the animation digraph finishes, the user may work with a different animation digraph or leave the interface.

A simple scenario demonstrating the basic function of the interface is shown in Figure 1. In this scenario, four icons, "skank", are connected in a ring fashion and a fifth one, "Dude" is connected to only one of the first four. Each of the four icons represents a process that inputs an integer value, computes some integer function of the given value, and outputs the obtained result. The purpose of the fifth process is to start the animation digraph by supplying the first process with an initial value. Upon receiving the animation digraph,

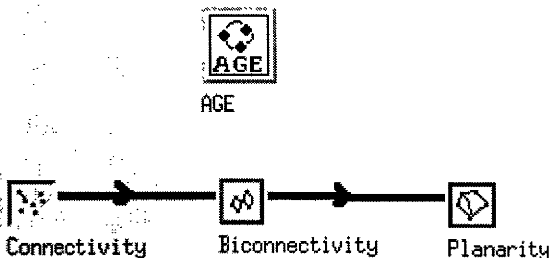


Fig. 2. A More Sophisticated Animation Digraph

the interface parses it, identifies the executable processes and constructs a communication network that will service the digraph. The user begins execution by selecting the fifth process. The fifth process sends a value to the first one which then computes an integer function of it and sends the result to the second process. This in turn sends its result to the third process and so on until progression reaches the first. The first receives the data value and continues with the loop. The loop cycles until the user terminates execution. All information passing and the sequencing of executing processes is handled by the interface

A more pertinent example of the tool's capability is demonstrated in Figure 2. In this scenario, a chain of processes (Connectivity, Biconnectivity and Planarity [5]) is linked to pass information which is displayed by an algorithmic animation environment. In this example the environment is AGE [1] AGE is a software environment for creating and interacting with visual displays of graph-theoretical concepts. Each AGE animation is considered an AGE client. In this case, the end user must supply an AGE graph to be used

with the Connectivity client. The Connectivity client finds the connected components of the AGE graph and passes them to the Biconnectivity client which finds the biconnected components. Biconnected components are passed to the Planarity client which tests each of them for planarity. When each of these clients takes control, its corresponding AGE results is displayed in the AGE window.

I.C Implementation

The prototype for this interface is created above Unix ^{TM1} on the X Window System^{TM2}. Unix was chosen because it provides a reasonably portable environment such that software can be integrated from one environment to another with only modest modifications[12]. Other operating systems may be capable of performing similar tasks as Unix but specific Unix functions are invoked by our current interface implementation. The X Window System is comprised of the X protocol, which interprets data streams from applications, and the X display server which performs the tasks requested by X clients. written using Xlib C routines [2]. The X Window System is intended to be portable like Unix thus promoting its use in a heterogeneous network of machines.

The interface was constructed using the OSF/Motif Toolkit^{TM3} which is layered on top of the X Windows platform. Motif is a very popular and well known interface toolkit which helps create easy to identify interface objects and tools. The large Motif object library was instrumental to the creation of some of the more sophisticated tools and objects of the interface.

[1]UNIX is a registered trademark of AT & T.

[2]The X Window System is a registered trademark of the Massachussetts Institute of Technology.

[3]Motif is a trademark of the Open Software Foundation.

Through this work, it became clear that object oriented techniques are the logical choice to express the relationships among the different objects this interface manipulates. Thus C++ was the natural selection of language. The object oriented nature of the Motif Toolkit blended in naturally with the C++ classes of the interface. The interface was implemented on Sun SPARC Stations^{TM4}. If the interface is desired to execute on other platforms, the code will need to be re-compiled. The conditions for re-compilation include the Unix platform, and the X Windows and Motif libraries.

I.D Related Work

The main objective of this work is to provide an interface that helps the user to integrate previously created algorithm animations and animation independent applications easily. An effective demonstration on algorithm animation creation is provided by AGE developed by Abello, Sudarsky, Veatch and Waller [1] and John Stasko's Tango Algorithm animation systems [10]. Algorithm animations are broken into three components: the algorithm component, the animation component, and the mapping component. By creating an editor for each of these components, one is able to create animations in a reasonable manner. A similar conceptual approach is taken by Sudarsky [11]. She provides a library of algorithmic animation primitives to help users program algorithm animations quickly and easily. Both AGE and Tango are influenced by a similar system called Balsa II by Brown[3]. Balsa II, one of the first systems to illustrate algorithms, provides a dynamic, interactive environment that helps to display a wide range of algorithms and data structures for animation.

From the user's point of view, a graphical interface as we present it in the following

[4]SPARC Station is a registered trademark of Sun Microsystems, Inc.

section behaves as a graph editor, where the vertices correspond to previously defined applications and the edges correspond to the flow and progression of data among them. Several questions concerning graph editors are addressed by the *EDGE* system created by Paulisch and Tichy [9]. Some of the issues tackled are automatic graph layouts, graph abstractions, adaptability and persistence of graphs.

Modularity is regarded as an essential design principle for graphical software packages. Thus, several graphical interface creation tools have been examined for guidance. The *Garnet* system, created at Carnegie Mellon, manipulates high and low level interface tools [7]. The important low level tools consist of a prototype-instance object oriented programming system, a constraint system, a graphical object system and an input handling system. Other efficient interface design tools are *Interviews* created by Linton, Vlissedes and Calder [6] and *Forms* created by Marc Overmars [8].

A good example of how an interface, like the one we are to propose, is to look and feel is the *Explorer* environment created by Silicon Graphics [4]. The interface tools provided are a distributed execution map, a map editor, a module or process librarian, and a datascribe where the user can control the format of each module's input and output.

One of the main algorithm animation tools worked with is the AGE environment [1]. AGE is an effective distributed animated graph environment. It is process oriented and built of multiple processes [14]. It makes effective use of interprocess communication. The work done by Sudarsky [11] provides a set of algorithmic primitives that can be used as building blocks for larger animations and algorithms. Any interface that integrates and controls such algorithmic networks must lie conceptually somewhere in between the algorithmic primitives and the animation system.

The entire development of AGE and this proposed interface has been done in the *Laboratory for Algorithms Design*(LAD) in the Computer Science Department of Texas A&M University. AGE is currently being used as an instructional tool for several graduate and undergraduate classes. Amongst being part of the research being done at the LAD, AGE acted as the testbed for this interface development.

I.E Thesis Outline

The next two chapters are dedicated to the description of the external and internal views of the interface.

In the first part of chapter II, the reader is taken through a typical interactive session that goes through the main steps from the access of the interface to its termination. The remaining of chapter II describes the administrative view of the interface for those who wish to register new applications with the interface.

Chapter III contains a top down description of the three levels of internal interface communication.

Chapter IV includes a discussion on the main lessons learned during the development and implementation of this interface. Future interface enhancements are also proposed in this chapter.

For completeness, we have included several appendices. The first appendix is the user's guide to the interface. Full descriptions of the interface tools and behaviors are included. The second appendix contains the specific details of application registration with the interface. A template and requirements for I/O Masters are included. The last appendix contains a glossary of terms introduced and used in this thesis.

CHAPTER II

INTERFACE EXTERNAL VIEW

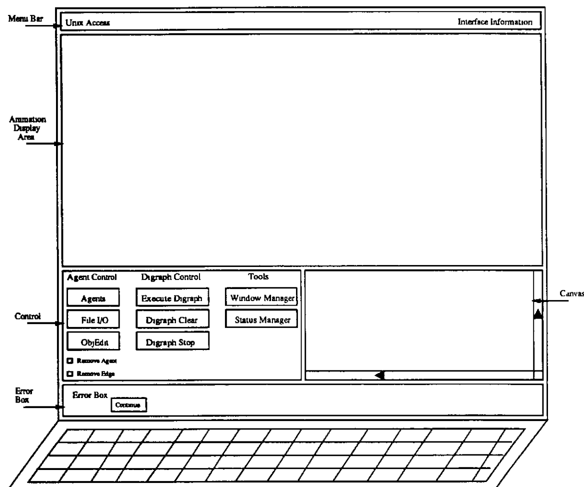


Fig. 3. Interface Appearance Upon Invocation

Before describing how the interface works internally, it is necessary to describe how it looks and behaves externally, (Figure 3). The first portion of this chapter goes through a typical interactive session. The reader is taken through all of the steps from accessing the interface to the termination of a session. The sample session demonstrates the behavior of

the interface and gives the reader enough information to construct his or her own animation digraphs and execute them. A full user's manual is supplied in the Appendix.

The second portion of the chapter describes the administrative aspects of the interface. The information needed by the interface to manipulate an application is discussed. Descriptions of registration tools are also given.

II.A Interface Access

We assume the user is logged onto a computer that is running under the Unix operating system, and is operating inside the X Windows environment. From a command interpreter window, the user can change the current directory to the directory where the interface executable resides. The Interface is invoked by calling the executable name "IntApp". After a few seconds of processing, the interface window should appear and look similar to Figure 3.

The interface is laid out in a simple, easy to use fashion. It is broken into five visual components: the Menu Bar, the Animation Display Area, the Control, the Canvas and the Error Box.

II.B Sample Session

To demonstrate the behavior and ability of the interface, we show how an animation digraph is created, executed and terminated. AGE behaves as the algorithm animation environment. In this example, the AGE clients composing the animation digraph are Connectivity, Biconnectivity and Planarity. Execution of the animation digraph starts with the Connectivity client.

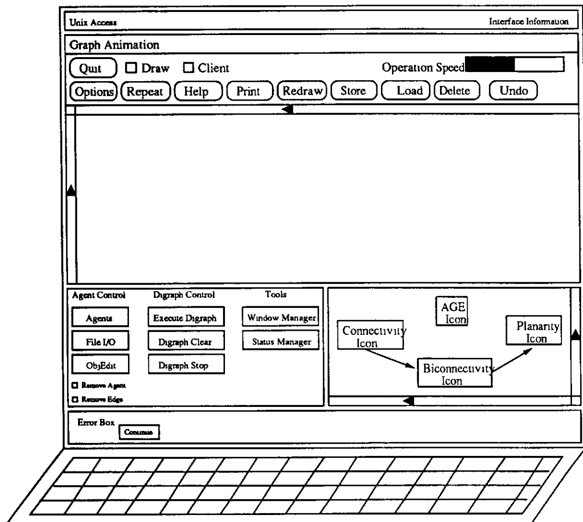


Fig. 4. Interface with AGE Animation Digraph

Connectivity is an AGE client that receives an AGE Graph, finds all of its *connected components* and then displays the first to the AGE window. A connected component is a subgraph such that for each pair of vertices, v and w, within the subgraph, there exists a path from v to w. The connected components are sent to the Biconnectivity client where the *biconnected components* are found. A biconnected component is a connected subgraph that does not contain any vertex whose removal will disconnect the graph. Each biconnected component is then sent to the Planarity client. The Planarity client takes a biconnected

component and tries to imbed it on the screen in such a way that no two edges intersect. The animation terminates when the execution of the planarity client is completed. The interface indicates termination by stopping the animation digraph icons.

In what follows, the reader is shown how to generate an animation digraph that looks similar to the one displayed in the Canvas subwindow of Figure 4. Once the animation digraph is created, the reader is shown how to execute it. The results of the execution are included. After the animation has concluded, the user may choose to terminate the animation digraph and leave the interface.

II.B.1 Animation Digraph Generation

There are two steps to generate an animation digraph: process invocation and interprocess connection. To invoke a process, the user presses the "Agents" button in the Control subwindow and choose either the "Environment", "Client" or "Executable" selection with the *right* mouse button from the new menu that pops up. Upon selection, the user is given an *Exec Box* to choose an available machine and application. Once theses choices are made, the user hits the "Execute" button in the Exec Box. The Exec Box disappears and after a few seconds, an icon for the application process appears in the Canvas subwindow. In our case, the icons correspond to AGE, and to Connectivity, Biconnectivity and Planarity clients. An icon can be moved within the Canvas by pressing it with the left mouse button and dragging it to a new location.

Execution of the AGE application is achieved by hitting the "Agents" button followed by the choices of "Environment", "Machine", and "AGE" and then hitting the execute button. The AGE window appears in the Animation Display Area and the AGE icon appears in the Canvas. The user then moves the AGE icon to the upper part of the Canvas. Once AGE

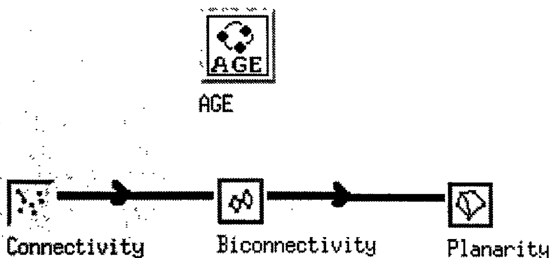


Fig. 5. A Sample AGE Animation Digraph

is invoked, the user invokes the Connectivity, Biconnectivity and Planarity clients. This is done in the exact same fashion as described above except "Client" is chosen instead of "Environment". The user may space out the icons in the Canvas for visual clarity.

Once the application processes are invoked, the user creates the process connections. A connection is made by pressing a source icon with the middle mouse button. The mouse pointer is then placed over a destination icon which is pressed with the middle mouse button. A directed edge appears and connects the source icon with the destination icon. In this example, the user creates connections from Connectivity to Biconnectivity and from Biconnectivity to Planarity. The animation digraph is complete and it looks similar to the animation digraph in Figure 5.

II.B.2 Animation Digraph Execution

Once the animation digraph is generated it can be executed. A process is set as the initial process by selecting the corresponding icon with the left mouse button. The initial process

icon frame then turns yellow. The User presses the Connectivity icon to initialize the animation digraph. The animation digraph can now be executed by pressing the "Execute Digraph" button in the Canvas subwindow.

Before allowing the Connectivity client to execute, the AGE server will query the user for an AGE graph. The user presses the AGE message area with the right mouse button selects "Done", creates an AGE graph and then hits the "Graph Entered" button. After this sequence of user actions, the Connectivity icon starts to animate and the Connectivity client starts in AGE. To operate the Connectivity client, the user hits the "Connectivity" button in the AGE window. When the client is complete, the AGE "Show" button is pressed. With that sequence, the Connectivity icon stops animating and the Biconnectivity icon starts. To start the Biconnectivity client in AGE, the user presses the "Biconnectivity" AGE button. When finished, the user presses the "Show" AGE button. This will stop the Biconnectivity client and start the Planarity client Pressing the "Planarity" button in AGE executes the algorithm. When the algorithm finishes, the user cycles through the AGE clients by pressing the round arrow AGE button until the word "IO Master" appears The user presses the AGE button that says "Algorithm Complete" The planarity Icon stops animating and the animation digraph is complete.

II.B.3 Animation Digraph Termination

Once the animation digraph is complete, it can be terminated by pressing the "Digraph Clear" button in the Control. The icons disappear from the Canvas as well as the AGE window from the Animation Display Area. The user may leave the interface by pressing the phrase "Unix Access" on the Menu Bar and pressing the "Return to Unix" button. The interface shuts down and the user is returned to the X window shell

II.C Administrative View

Although not a large part of the external view of the interface, the application administration plays a vital portion in the expandability of the interface. The registration of new applications is not a difficult procedure but one that must done with thought and care. Errors may have direct effects on how an application behaves and how it interacts with other processes.

II.C.1 Needed Information

For every application that executes on the interface, there is a set number of informative data elements describing the application and communication behavior. Data concerning the name and executable are needed to launch the application correctly. Bitmap information is needed to display the appearance of the icon in the Canvas. Input and output demands are needed to inform the interface how it communicates with other processes.

For the interface to launch an application properly, the application's name and executable path are required. If the interface tries to launch an application that does not exist at the given path, an error will be displayed in the Error Box. The interface will continue making the error if not corrected. The "type" of application is also needed. The interface accepts three different types of applications: Animation Environments, Clients and general executables. If the application is an animation environment, the path of an I/O Master is needed for it. I/O Masters are programs that communicate between algorithm animation environments and the interface, and are described in full detail in chapter III.E. Clients are considered as applications who need an Animation Environment to operate. Clients are prevented from launching until an Animation Environment is provided. General executa-

bles refer to any applications that are independent of environments or clients. Executables have no constraints on launching.

Every Application has a set of *bitmaps* displayed within its icon. Each icon is broken into two animation modes: executing and stationary or non executing. The bitmaps are cycled to give the icon an animated appearance. The bitmaps used for the execution mode animation do not have to correspond to the bitmaps used in the non-executing animation mode. Each bitmap is stored, individually, in the standard X Window bitmap format [2]. The bitmaps may be created using the X Window bitmap editor *bitmap*.

Each application needs to define the necessary input sequences needed to execute. There may be multiple acceptable input sequences for each application. An example is a client that needs a graph or a filename to execute If the client receives the filename instead of the graph, it can read from the contents of the inputed filename. Each necessary input sequence is defined by integer values, floating point values, words, graphs, file names, environmental specific data such as AGE graphs, and matrices. The output sequence for an application is defined in the exact same fashion as the input sequences The input and output sequences are recorded in a header file for each application. Instruction on the creation of header files in included in Appendix B.

II.C.2 Techniques of Registration

All information defining an application, except for the input and output sequences, is stored in files called *Catalogs.* The Interface uses a default catalog loaded during startup time. The user may load Catalogs by using the *Catalog Loader* which is accessed via the Control button "File I/O". After the button is pressed, a menu appears giving the options of "Catalog" or "Digraph". These buttons cascade to further options of "Load", "Save"

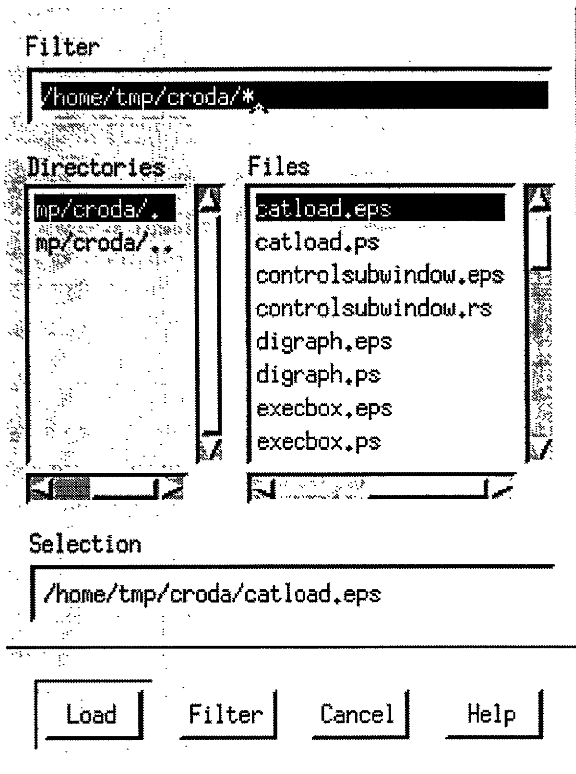


Fig. 6. Catalog Loading Menu

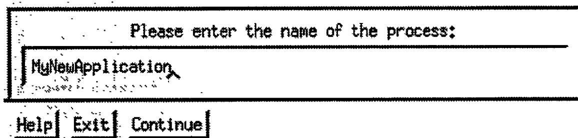


Fig. 7. Query Box

or "Cancel". The button sequence "Catalog → Load" produces a loading menu called *CatLoad*, see Figure 6. This tool enables the user to select a catalog and load it into the interface.

If the user chooses, he may create his own catalog by using a file editor and suplying the needed information to a catalog file. The format and contents of a catalog are described in Appendix B. The user may also define a new application by pressing the "ObjEdit" button in the Control subwindow. Upon pressing the button, a window appears called the *Query Box* which prompts the user for information and then saves the results to the current Catalog, see Figure 7. The questions in the Query Box are self explanatory and the user always has the option of exiting the procedure. The information taken by the Query Box will be sufficient to append to the catalog file and create a new header file.

CHAPTER III

INTERNAL DESCRIPTION

III.A Global Interface View

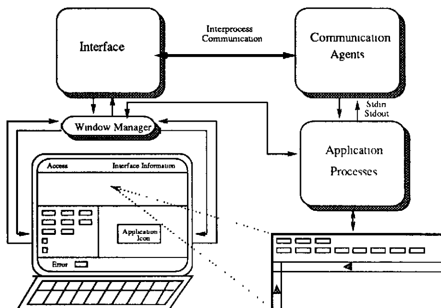


Fig. 8. Global Interface View

If we remove ourselves from preconceived ideas about operating systems and window managers, we can derive a global view of the internal implementation of the interface. What is minimally needed to implement an interface such as this is an environment that allows us to obtain the functionality demonstrated in Figure 8. On the highest level, there is a terminal with an interface view displayed on it. The images on the terminal are driven by a window manager which receives requests from an application. This application controls the behavior of the images on the terminal and communicates with other processes called *communication agents*. Communication agents control the behavior of other appli-

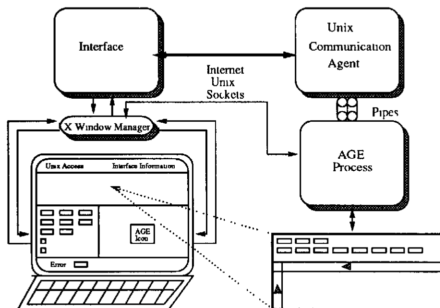


Fig. 9. AGE Interface View

cation processes working with the same window manager. The application processes have no knowledge of the agents controlling them and pass information to and from them via understood file descriptors.

We have been able to achieve this on top of the Unix operating system and the X Windows environment. Execution of the interface depends heavily on these environments. Our actual implementation of Figure 8 looks more like Figure 9. The X Window system is essential for the driving of animated icons and remote displays. Unix interprocess communication techniques like internet sockets and pipes are essential to the interface. Other intrinsic functions in Unix such as *fork* and *exec* play a large roll in the behavior of the interface. Because of the interface's heavy dependence on both these environments, Unix and X Windows must be available for our implementation of the interface to operate successfully.

III.B Levels of Interface Communication

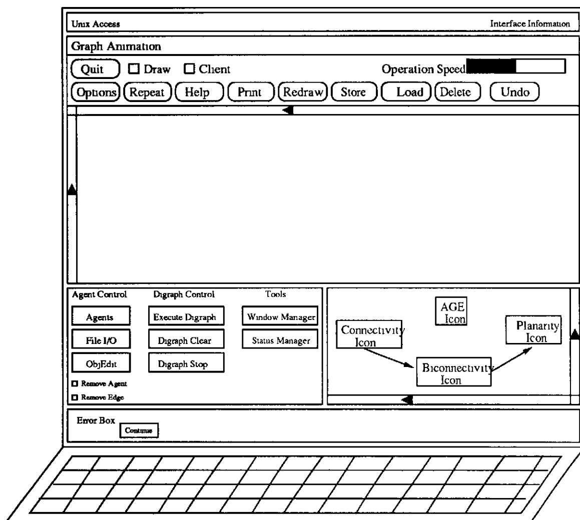


Fig. 10. Interface With AGE Animation Digraph

Figure 10 is a copy of Figure 4 from chapter 2. The sample animation digraph appearing in the Canvas section will help to describe the internal view of the interface because its execution requires the activation of the three levels of internal interface communication. Figure 11 is the corresponding internal view of the interface for the external view of Figure 10. The first level occurs within the interface itself. The user sends events to the *User Event Handler* via the X Window Manager which is managing the interface display on the

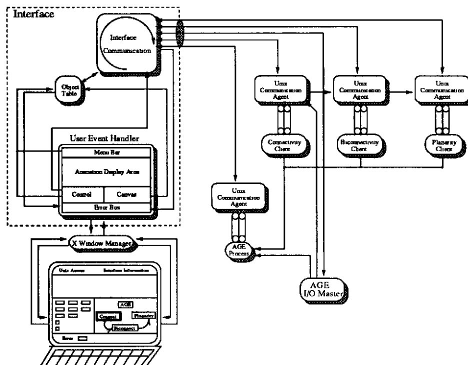


Fig. 11. The Three Levels of Interface Communication

terminal screen. The events are interpreted and messages are sent to the other internal mechanisms of the interface to be acted on.

The second level deals with the communication between the interface and the *Unix Communication Agents* (UCAs). The UCAs are spawned processes from the interface which communicate with it via Unix internet sockets. Sockets are a facility of Unix that provide two way flows of data, across the internet, usually between two processes. The UCAs spawn the application processes sent as parameters during their execution. The UCAs communicate to the application processes through standard input and standard output via Unix pipes. Hereafter, we will refer to these as *Stdin* and *Stdout*, respectively. A pipe is a facility of Unix that provides one way flow of data usually from one process to another on the same machine. UCAs receive commands from the interface to execute or stop the

application processes or make communication links to other Unix communication agents. These links are implemented by Unix internet sockets. Received data from other agents is relayed to the application process through its Stdin. The process' Stdout is received by the agent and distributed to other agents.

The third level involves all of the previous levels plus a new entity called the *I/O Master*. In this application, the current animation environment is AGE thus the new element is the AGE I/O Master. The I/O Master acts as the link between the animation environment and the interface. The interface spawns the I/O Master and communicates to it through Unix internet sockets. The I/O Master connects to the animation environment and communicates to it through the environment's communication apparatus. In this example the medium is Unix internet sockets. The I/O Master receives commands from the interface to get data information from the animation environment. The environment data is relayed to the communication agent of the client currently attempting execution. The communication medium between the I/O Master and client's communication agent is Unix internet sockets.

III.C The First Level of Interface Communication

The first level of interface communication involves the interface itself and its external connections to other computational entities such as the X Window Manager and UCAs, see Figure 12. The purpose of the interface is to provide a simple, graphical method for interactively working with algorithm animation systems, the algorithm animations associated with them and any other executables that may interact with the algorithms. Interelated animations, represented here by animation digraphs, are interactively created, manipulated, executed and terminated. Animation digraphs consist of vertices and directed edges. The

vertices are animated icons that represent specific application processes. The edges correspond to lines of communication between two processes and the sequences of execution. The interface interprets an input animation digraph, activates the corresponding computational chores associated with each icon node of the digraph, and provides the necessary structures to guarantee the assigned task is to be completed successfully or be aborted gracefully in case of abnormal conditions.

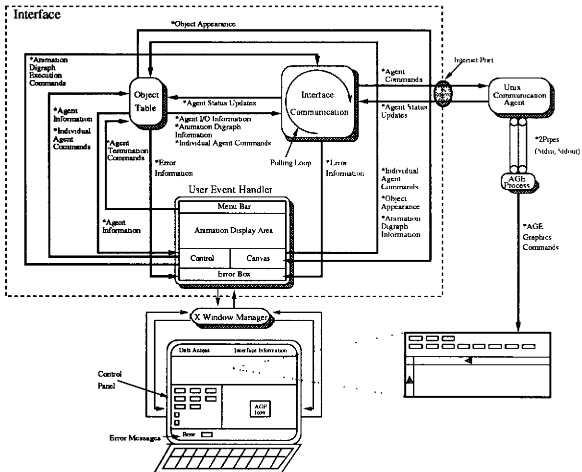


Fig. 12. The First Level of Interface Communication

The interface achieves these tasks by its three internal mechanisms: the *User Event Handler*, the *Object Table* and the *Interface Communication*. The User Event Handler communicates with the X Window Manager and receives all visual events supplied by the

user. These events are parsed into commands and distributed to either the Object Table or the Interface Communication.

The *User Event Handler* sends individual agent commands to the Object Table along with agent information, animation digraph information and object appearance information. The objects referred to are the icons within the animation digraph. The Object table sends back to the User Event Handler agent information, object appearance information and error information.

The *User Event handler* sends the animation digraph execution commands to the Interface Communication. The Interface Communication returns error information.

The *Object Table* sends the Interface Communication messages concerning agent I/O information, animation digraph information and individual agent commands. The Interface Communication returns agent status updates.

The *Interface Communication* acts as the communication engine for the interface with the Unix Communication Agents and the I/O Masters. Agent commands and agent status updates are sent to and received from Unix Communication Agents respectively. Client information and needed client data are sent to I/O Masters while animation environments statuses are returned.

III.C.1 The User Event Handler

The *User Event Handler* is designed to act as the front end control mechanism for the interface. Its purpose is to establish communication with the the X Window Manager and receive all visual events captured by the manager pertaining to the interface, see Figure 13. These events are parsed and delivered to their respective mini-handlers within the mechanism. The mini-handlers translate the events to interface commands and send them

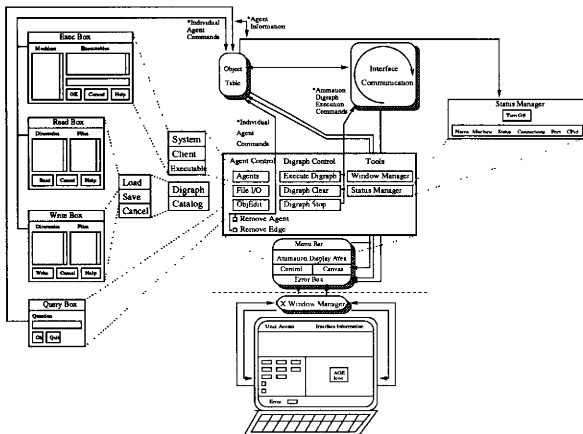


Fig. 13. Internal Interface Message Flow

off either to the Object Table or to the Interface Communication.

The User Event Handler performs these duties by its five internal *mini-handlers*: the *Menu Bar*, the *Animation Display Area*, the *Control*, the *Canvas* and the *Error Box*. Each of these mini-handlers drives their corresponding subwindow in the interface display. There is no direct communication between any of these mini-handlers as they behave as independent entities and communicate to the rest of the interface through the Object Table or the Interface Communication.

The *Menu Bar mini-handler* drives the menu bar subwindow of the interface. It receives events that tell it to return to Unix, or display informational windows concerning interface usage or interface version. The return to Unix event initiates a series of agent commands

that are sent to the Object Table to order the termination of the application processes and communication agents. The Menu Bar then shuts down the interface.

The *Animation Display Area mini-handler* drives the animation display area subwindow of the interface. Since this submenu blocks out space on the screen that is to be used by an animation environment, no events are sent to the handler. Thus no commands are originated either.

The *Control mini-handler* drives the Control subwindow of the interface. It breaks up the subwindow into three subsections: Agent Control, Digraph Control and Tools. The "Agent Control" section contains three buttons and two toggles. The "Agents" button generates a menu which in turn generates one of three Exec Box windows corresponding to a system, client or executable. The Exec Box generates an individual agent command which is sent to the Object table. This command instructs the Object Table to launch an agent for the selected application on the selected machine. The "File I/O" button generates a series of menus which terminate in either a Read Box or a Write Box. These boxes generate individual agent commands sent to the Object Table. These commands tell the Object Table to launch specific agents and to save digraphs or catalogs with the given filenames. The "ObjEdit" generates a Query Box. The Query Box sends to the Object Table agent information received as input from the user. The "Remove Agent" and "Remove Edge" toggles send individual agent commands to the Object Table describing which agent or edges are to be removed.

The "Digraph Control" section contains three buttons. "Execute Digraph", "Digraph Clear", and "Digraph Stop". Each of these button events generates animation digraph execution commands which are sent to the Interface Communication. These commands

Turn OFF

#	Name	Machine	Status	Connections	Port	CPid
0:	Dudel	sparc50	Armed	skank : sparc50	1472	5619
1:	skank	sparc50	Stopped	skank : sparc73	1474	5623
4:	skank	sparc73	Stopped	skank : sparc75	1176	11342
3:	skank	sparc75	Stopped	skank : sparc75	1156	6543
2:	skank	sparc75	Stopped	skank : sparc50	1157	6544

Fig. 14. Interface Status Manager

tell the Interface Communication to start executing the digraph, remove all the agents and application processes of the digraph and to stop the executing digraph, respectively.

The "Tools" section contains two buttons: "Window Manager" and "Status Manager". The "Window Manager" button is desensitized or not active and receives no user events. The "Status Manager" button generates the Status Manager Window. The Status Manager window receives agent information from the Object Table for every object on the subwindow and then displays the information in the window, see Figure 14.

The *Canvas mini-handler* drives the canvas subwindow of the interface. The Canvas displays the animation digraph and provides facilities for the user to modify the digraph and control some of the individual application processes. The Canvas sends to the Object Table messages containing object appearance and animation digraph information. It receives from the Object Table object appearance information. This information is used to update the appearance of the icons in the digraph. When the user wants to manipulate the execution state of an application process from the Canvas, the Canvas sends an individual agent command to the Object Table. This command contains the updated state information.

The *Error Box mini-handler* drives the error Box subwindow in the interface. This

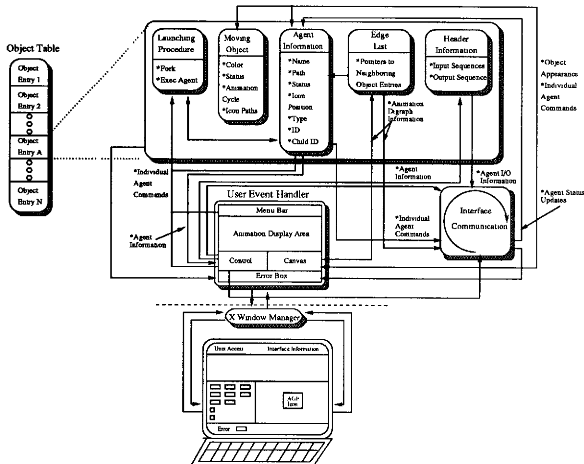


Fig. 15. Object Table Entry Information Flow

handler receives error information from the The Object Table and the Interface Communication. Errors are displayed in the box and removed when the user hits the "Continue" button. The Error Box sends or receives no other information.

III.C.2 The Object Table

The Object table acts as the global data base for the interface, see Figure 15. Aside from storing information about animation digraphs and its individual elements, it also has certain procedures associated with it. The purpose of the Object Table is to receive and record information about elements of an animation digraph, supply information to

the Interface Communication and provide the launching procedure for the communication agents. Much of the information sent to the Interface Communication involves individual agent commands. The Object Table is actually a dynamic array of object entries. There is one object entry in the Object Table for every vertex within the animation digraph.

The Object table achieves all of its duties by parsing the messages and commands into five groups: the *Launching Procedure*, the *Moving Object*, the *Agent Information*, the *Edge List* and the *Header Information*. Only the Launching Procedure and the Agent Information groups share information. The contents of the Edge list are lent to the Agent Information whenever it is sending agent information to the Control

The *Launching Procedure* receives all the individual agent commands that request the launching of a Unix communication agent The Procedure accesses all of the needed information from the Agent Information, forks a process and execs a new Unix communication agent with the information needed to launch the new desired application. The process id of the agent is sent to the Agent Information

The *Moving Object* group receives object appearance information from the Canvas mini-handler. Information dealing with the appearance of the icons in the animation digraph is stored here. This information includes the icon frame color, icon status, icon animation cycle and the paths of the bitmaps that compose the icon animation.

The *Agent Information* group receives individual agent commands and agent information from the Control mini-handler and returns to it agent information. Before sending the agent information to the Control, the list of neighboring objects must be borrowed from the Edge List. From the Canvas, it receives individual agent commands and returns nothing. All of the received individual agent commands are interpreted and sent to the

Interface Communication. The group receives agent status updates in return. Much of the information stored in the Agent Information, such as name path and type, is read in from the default catalog during the interface startup (see section II.C.2).

The *Edge List* keeps pointers to all the object entries in the table that are neighbors to the entry in the animation digraph. It receives the animation digraph information from the Canvas mini-handler and sends the information out to the Interface Communication. The neighboring information is also given to the Agent Information whenever it is sending agent information to the Control mini-handler.

The necessary input sequences and the output sequence for the corresponding application are stored in the *Header information group*. The agent information is received from the Control mini-handler and sends the sequence information off to the Interface Communication.

III.C.3 The Interface Communication

The *Interface Communication* acts as the central headquarters for all information that is sent over the internet, see Figure 12. Its purpose is to be a gate for all information entering and leaving the interface that is not involved with the window display. All agent commands are broken down into their individual components and distributed to their proper destinations. The Interface Communication gets information from the Object Table to satisfy the requests of a UCA. An example is the input and output sequences for a specific application. The agent requests this information from the Interface Communication upon registration. The Interface Communication gets this information from the Object table. The Interface Communication performs its function by remaining in a polling loop with its socket and all of its connected sockets, checking each for new messages or connection

requests. As an agent connects with the Interface Communication, certain information about the agent is kept and placed in a list of all the communication agents before the status update information is sent to the Object table. When an agent command is received, the destination for the command is known immediately.

The animation digraph execution commands, received from the Control mini-handler, are broken down into individual agent commands and are distributed to their corresponding communication agents in sequential order. All individual agent commands from the Object Table are translated to their basic components and sent to their communication agent. The Interface Communication receives status updates from the communication agents and that information is passed on to the Object Table. All error information from the Interface Communication or an Agent is sent to the Error Box mini-handler

III.D The Second Level of Interface Communication

III.D.1 Unix Communication Agent (UCA)

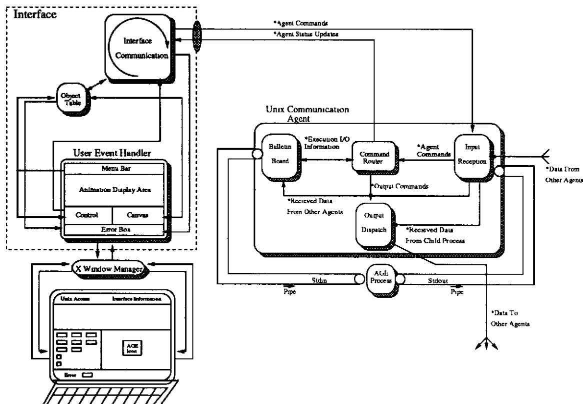


Fig. 16. The Second Level of Interface Communication

Every application process executing within the interface animation digraph is connected to a Unix Communication Agent (UCA). When the user invokes an application from the Control subwindow, a command gets sent to the Object Table to fork the process and exec a UCA with the necessary parameters to launch the desired application. When a process *forks* in Unix, a copy of the process is made and starts executing. The process that did the fork is considered the *parent*, the resulting process is the *child*. When a process does an *exec* Unix system call it transforms itself into a new program. Upon execution, the UCA connects with the interface and sends information about itself such as name and

process id. Once the registration is complete, the UCA forks the process and execs the desired application. Before the exec, the UCA ties the child's standard input (Stdin) and standard output (Stdout) to itself via two Unix pipes. Stdin refers to the default stream a process reads from if no specific file is supplied. Stdout refers to the default stream a process writes to if no specific file is supplied. The child process' default input and output go directly to the parent UCA. The purpose of the UCA is to launch the application process, receive and execute interface commands concerning the application, perform the internet communication for the application process and report the process' status back to the interface.

The UCA's duties are allocated to four internal mechanisms: the *Bulletin Board*, the *Command Router*, the *Input Reception* and the *Output Dispatch*, see Figure 16. As a unit, the UCA remains transparent to the application process yet interacts with the main interface and other applications. No modifications are made to the application process other than the routing of its communication through its Stdin and Stdout.

Internally, the Input Reception receives all communications from the interface, the child process and all other UCAs. The Input Reception signals the Command Router when there is a message from the interface, sends data from the child process to the Output Dispatch and data from other UCAs to the Bulletin Board. The Command Router receives agent commands from the interface and returns status updates. It exchanges execution I/O information with the Bulletin Board and sends output commands to the Output Dispatch. The Bulletin Board sends output to the application process via the pipe connected to the child's Stdin. The Output Dispatch sends data to all other connected UCAs.

III.D.2 Command Router

The purpose of the *Command Router* is to act as the central decision maker for the UCA. The socket to communicate with the interface is owned by the Router. Although the Input Reception monitors the socket for any new messages, a signal is sent to the Router from the Input Reception informing that a message from the interface has been received. To fulfill its functionality, the Command Router parses and executes all interface commands, launches, executes and stops the child application process, sends the input and output sequences to the Bulletin Board, decides if the application process is allowed to execute based in its received information from the Bulletin Board and sends output connection commands to the Output Dispatch, reflecting the connection commands sent by the interface.

III.D.3 Bulletin Board

An essential mechanism to the UCA is the *Bulletin Board* mechanism, see Figure 17. The purpose of the Bulletin Board is to act as a data storage bank for all received data until the data is needed. Because it keeps track of all received information, it judges if the available data is sufficient to satisfy the needed input sequences. When the application process is allowed to execute, the Bulletin Board send its information to the child process via its Stdin pipe.

The Bulletin Board achieves these duties with the help of its three internal mechanisms: the *Message Parser*, the *Input Reception* and the *Data Information Queues*. The Message Parser receives all data sent to it from the Input Reception and places each data element into the appropriate queue of the Data Information Queues. The Queue information, such as size, is sent to the Execution I/O Information mechanism. The execution I/O Information

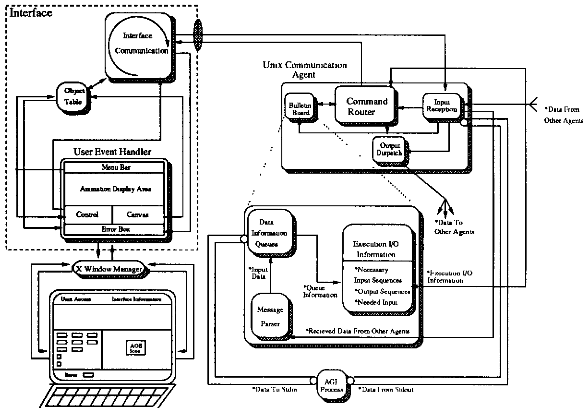


Fig. 17. UCA Bulletin Board

stores the input and output sequences from the Command Router and returns a YES/NO execution decision.

III.D.4 Input Reception

The *Input Reception* monitors all lines of external communication and sends the received data to its proper place. It has a special relationship with the Command Router as it monitors the Command Router's socket to the interface and signals when a message arrives. The Input Reception does not read the socket. The Input Reception blocks the execution of the UCA until information arrives. Once the information arrives, the Input Reception reads in the information and activates the receiving mechanism with the inputed data.

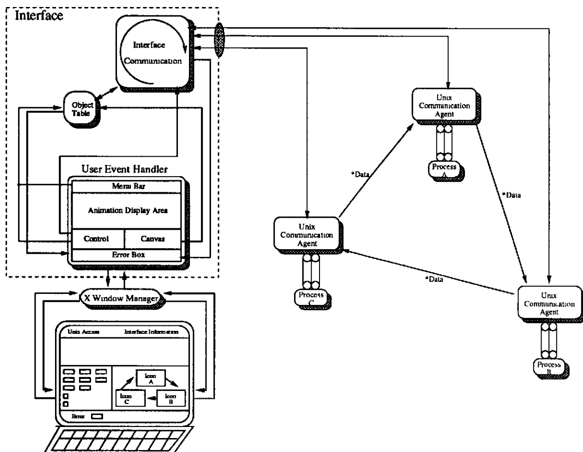


Fig. 18. Multiple Process Interaction

III.D.5 Output Dispatch

The purpose of the *Output Dispatch* is to relay all information received from the child application processes from the Input Reception to all connected UCAs. The Output Dispatch creates lines of communication to those UCAs indicated by the Command Router.

III.D.6 Multiple Process Interaction

An example of the UCA's potential is demonstrated by the multiple process interaction described in Figure 18. In this scenario a ring of applications have been created in the interface and internally the UCAs have been interlinked with each other. When the interface

sends the execution command to the UCA of process A, process A is allowed to execute. The output of process A gets read by its UCA and sent to the UCA of process B. The UCA interprets the output action of A as a signal of completion and tells the interface that process A is complete. The interface sends the execution command to the UCA of process B which executes the process and sends it the data received from A's UCA. In the same sequence of events, process B executes and the data is sent to the UCA of process C. Process B stops, process C executes and data is sent to process A's UCA. Depending on the nature of process A, as soon as the UCA of A receives the command to execute and the data from C, process A will execute, continuing the loop. The loop will continue executing until the interface stops execution or one of the processes stops and terminates naturally.

III.D.7 File Input and File Output

The relationship between the UCA and its child application process is exploited by the FileReader and FileWriter programs. Example digraphs demonstrating their behavior are shown in Figure 19. In this example, process A receives data from the FileReader and process B sends data to the FileWriter. During execution, the FileReader is allowed to execute before process A. The FileReader needs no input so it starts to execute immediately. The FileReader sends a request to its Stdout for the necessary input sequence of process A. The UCA receives the request and relays it to the interface. The interface returns the sequence to the UCA which sends it to the FileReader via its Stdin. With the help of a Read Box, the FileReader opens an input file, reads the data according to the received sequence then sends the data to its Stdout which is received by the UCA and sent to the UCA of process A. The FileReader stops execution and process A starts reading the data from its Stdin.

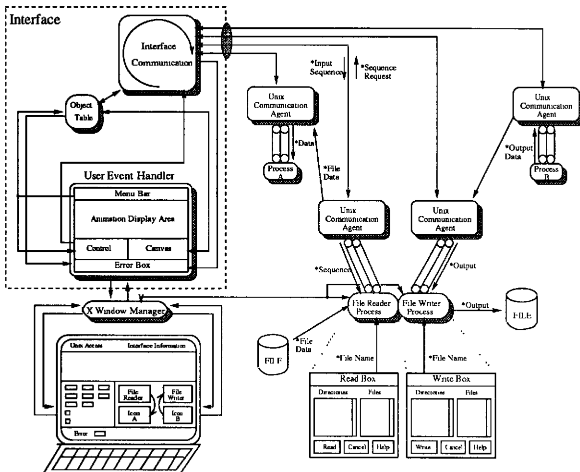


Fig. 19. File Input and File Output

In the other example, process B is allowed to execute before the FileWriter. B outputs data to its Stdout and is read by its UCA. The data is sent to the UCA of the FileWriter which sends it to the FileWriter With the help of a Write Box, the FileWriter opens a file and outputs all information sent by its UCA.

III.E The Third Level of Interface Communication

III.E.1 I/O Master

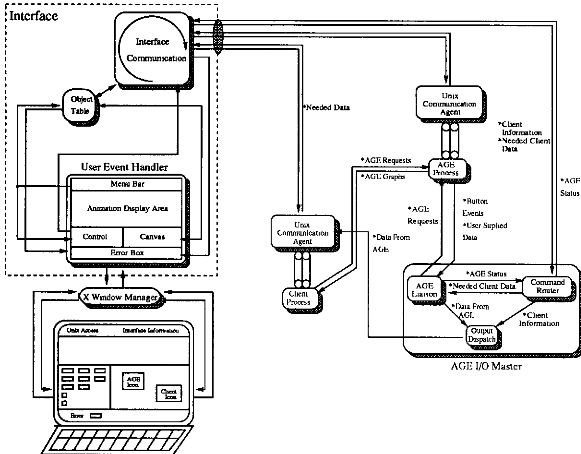


Fig. 20. The Third Level of Interface Communication

Before the first client is invoked from the interface, the I/O Master for the particular animation environment is launched. The I/O Master is spawned and connects back to the interface and connects to its corresponding animation environment. The I/O Master is confined to using the animation environment's communication methods. The I/O Master's purpose is to link the animation environment with the main interface. When the client process is set to execute, the interface queries the client UCA for the data it needs to

execute. The interface sends the needed client data information to the I/O Master along with the connection information for the executing client. The I/O Master then creates a series of requests and sends them to the animation environment in order to prompt the user for the necessary client information. Based on the requests from the I/O Master, the animation environment obtains the user suplied data and sends it back to the I/O Master. The I/O Master then sends the information to the UCA of the executing client. The Client process should have the necessary input data to execute. If the client does not have a method to output any information to signal completion, the I/O Master places a device in the animation environment that can be invoked upon client completion. The I/O Master receives the completion event and sends it back to the interface to let the next client start execution.

The I/O Master is composed of three computational mechanisms the *Command Router*, the *Environment Liaison* and the *Output Dispatch* (Figure 20). The Command Router receives the interface commands along with the client information and the needed client data. The notification of special events are sent back to the interface. An example of such an event is the client completion signal. The Command Router sends the needed client data to the Environment Liaison and gets back environmental status information. It sends the client information to the Output Dispatch. The Environment Liaison sends requests to the animation environment and gets back user suplied data and special device events. The received data is relayed to the Output Dispatch where it is sent to the UCA of the client needing input data.

III.E.2 Command Router

The *Command Router* receives all communications from the interface and distributes the information to the Environment Liaison or the Output Dispatch. Its purpose is to communicate with the main interface and route the received commands to the proper destinations. Like the UCA Input Reception, the Command Router blocks the I/O Master until a message arrives from the interface. The message is immediately interpreted and sent to its proper channels.

III.E.3 Environment Liaison

The purpose of the *Environment Liaison* is to act as a client to the desired animation environment. The needed client data is translated into requests and sent to the environment. The results of the requests are sent back to the Liaison and relayed to the Output Dispatch. The Environment Liaison communicates with the animation environment according to the environment's methods.

III.E.4 Output Dispatch

The purpose of the *Output Dispatch* is to connect with the UCA of the client needing input and transmit data to it once the information arrives from the Environment Liaison. Unlike the Output Dispatch of the UCA, information is sent to only one connected UCA at a time. If the I/O Master retrieves information for the same client at a later time, it does not need to reconnect with that client's UCA.

CHAPTER IV

CONCLUSION

IV.A Results

This thesis proposes an interface that allows the end-user to integrate algorithm animations and independent applications. Its evolution has contributed to the development of several inter-related issues. The distribution of duties amongst internal mechanisms provides a good methodology for the construction of interfaces dealing with multiple communicating processes. The uses for this interface are diverse, ranging from research to teaching. This interface achieves its desired objectives and yet it is subject to limitations. These limitations provide insight for the implementation of future systems.

IV.A.1 Methodology

The internal allocation of tasks within the interface provides a good methodology which can be applied for similar systems dealing with control and interprocess communications of previously defined applications. Three internal and two external mechanisms provide a simple yet powerful way for the user to manipulate otherwise uncontrollable external applications.

The User Event Handler, Object Table and Interface Communication mechanisms contribute to a stable foundation for handling input and output. The interface receives input, in the form of window events by the User Event Handler. These events are translated into commands which are distributed as output by the Interface Communication. Dedicated to

the delivering of command output and the reception of processes reply, the Interface Communication provides a unique link to all process activity outside the interface. The Object Table provides a database for the recording of information provided by the user and process replies received from the Interface Communication. These three mechanisms create a triangle through which the user may communicate with external independent applications.

The Communication Agents act as the external arms of the interface While connected only to the Interface Communication mechanism, the agents perform the interface's commands externally. The Agents remain invisible to applications at all times as they translate the received commands from the interface. They remove the responsibility of transporting input and output from the interface. This freedom allows the interface to dedicate itself to the execution of user input.

Similar to the Communication Agents, the I/O Masters act as the external communication links from the interface to the otherwise uncommunicative animation environments. Because the I/O Masters are specialized applications that can speak to both the interface and a particular environment, the interface can use the environment to get data essential for execution, from the user.

IV.A.2 Uses

The needs of the user dictate how this interface is to be used. It has potential value in any situation where the user may need to set up sequences of applications that need to pass input and output with each other and execute in a desired order. This has immediate value to the research and educational communities. Other than the realm of algorithm animation, this interface has applications in the visualization of large scale numerical data systems and the visualization of distributed animation and simulation.

An important research application for this interface is to act as a testbed for distributed algorithms. All execution of animation digraphs start with just one process. That process may split execution into multiple processes. Thus after the termination of the initial application, control may be passed onto a number of independent processes executing in parallel. Distributed algorithms can be decomposed into their individual components, layed out in an orderly fashion as animation digraphs, and executed.

Another possible research application for this interface is the development and testing of sophisticated algorithms. We use the term sophisticated to refer to algorithms composed of other algorithms. The modular nature of the animation digraph allows the user to easily add, swap and remove applications from the animation digraph easily. Thus animation digraphs can be executed, modified and re-executed repeatedly until the desired results are achieved. This is a more efficient method for algorithm testing than the traditional one-program method.

An important teaching application for the interface is the demonstration of algorithm significance and function. Quite often a student may not perceive the utility of a particular algorithm as it is taught. However, as a unit in an animation digraph, the student may observe the relevance and usefulness of an algorithm as it interacts with others. An example of this has been displayed earlier in the animation digraph consisting of Connectivity, Biconnectivity and Planarity. In this example, a student may not comprehend the functional differences between the Connectivity and Biconnectivity algorithms, but when they are observed as elements of a sequence leading to a final destination, Planarity, the student can begin to understand the usefulness and functionality of these particular algorithms.

The visualization of large scale numerical data system can be aided by this interface.

By constructing animation digraphs consisting of numerical filters, researchers can visualize large quantities of data. Quite often programs are written to handle one type of input and to output another. With this interface, pipelines of these programs can receive and process information that terminate in one or many display windows. Meteorologists can use this interface as a tool for processing data received from storms in order to study phenomenon such as hurricanes. Petroleum engineers can take data processed from other machines, such as Crays, and reconstruct and visualize layers of oil deposits. In the same fashion, geologist can study the movements and changes in the Earth's crust. Astronomers can also be aided as large amounts of satellite data can be taken to reconstruct and visualize planetary surfaces.

The distributed nature of the interface opens new horizons in animation and simulation. Animations are typically one program entities that take one object, expose it to a series of forces and changes in environment and display the resulting activities. This interface allows the possibilities for many of these animations to communicate data with each other as they are driving their images to display. If the results of these animations can be composited to one display then we would have the effect of one animation under the influence off many different factors. Take, for example, an animation of a walking dog, an animation of a hurricane, and an animation of trees. Now run all these animation in parallel, letting them send information back and forth to each other, and displaying all the results to one scene. The result would be a dog trying to walk in a very bad storm while trying to react to the actions of the moving trees and the strong gusts of winds. This concept is still yet very abstract and still needs much thought. But the potential exists for further research.

IV.A.3 Limitations

One of the most significant limitations of this interface is the problem of applications created for other windowing environments other than X Windows. AGE is an example of such an application. AGE proved to be a challenge to integrate with the interface. The demands set by the Sunview windowing system became a difficult obstacle to work around. There is no guarantee there will be windowing environments to support all platforms. Hopefully over time, applications will be developed in a more uniform standard. If not, translators are essential to the universal communication and interaction of applications.

One of the conditions placed upon the developers of programs that are executed within this interface, is the restricted used of *Stdin* and *Stdout*. In order to communicate information with other processes, these two channels are dedicated to the transmission and reception of data. The developer is forced to use *Stderr* for the display of debugging messages and other information. User-supplied input must now be supplied through other other than Stdin. The sacrifice of these two channels was essential to provide an environment where predefined applications could interact with each other without having to go through major modifications.

Re-executability of animation digraphs is hindered by the fundamental nature of some applications. Unless the application has been written to stay within an internal loop, it may die or become static once it has produced output. This factor brings up many difficulties for the reusability of animation digraphs. Because each icon in the animation digraph represents one real process, that process may need to be recreated in order to function as a participating member of the animation digraph. Processes that have been stopped after output behave as dead weight, as their icon still appears in the animation digraph. The

responsibility for re-executable applications is then delivered into the hands of the developer.

The aforementioned limitations play roles in the registration process that all applications must go through. All applications must obey certain criteria to perform effectively within an animation digraph. While these criteria are not many in number, their pure presence is enough to decrease the utility of the interface to some degree. Thus, the removal of these limitations is presented as one of the future enhancements for the interface.

IV.B Future Enhancements

IV.B.1 I/O

As was mentioned in the first section of this chapter, this interface forces the user to execute under a certain limitations. Quite often the requirements for meeting these limitations require code modification. A future goal would be to minimize and possibly remove any need to modify the application code. A primary example of modification is the process signaling to the Communication Agent that it has completed its computation. Without this signal, the Communication Agent has no idea when the process has completed its tasks. This problem is compounded by the fact that the process outputs data only at the end of its computation. The outputting of data is the signal that the application has finished. The problem arises when the process is manipulating environment specific data like AGE graphs. In this situation, there is no need to output anything but an AGE graph. But since the clients are using the AGE server as a communication medium of AGE graphs, they still need to output something to the Communication Agent to signal its completion. In this instance, the signaling is done with transmission of a semaphore Other code modification is needed in the transmission and reception of data. The process has

to follow a specific sequence when sending and receiving the data. For both the signaling and data transmission problems, the user should not have to make any modifications to the source of the application. All applications should be ready to be used by the interface.

IV.B.2 Icon Window Modification

Currently, the icons of the animation digraph are viewed as an animated bitmap within a color-coded frame. The icon may change position and the bitmap may also change but otherwise the icon is static. A useful future extension to the interface would be the provision of tools to the user to modify the icon window. The size and color of the icon are two important attributes that are not alterable

Making the icon window an independent viewport is another interesting enhancement. Currently each icon is a very small X window. If this window could be used to display information from the application and control structures for the manipulation of input parameters, an entirely new avenue of icon manipulation can be created.

To supplement the creation of bitmaps, an animated bitmap editor provided by the interface would be helpful. Currently the user must use the X Window program "bitmap" to generate the icon bitmaps individually. An editor that allows the creation of the bitmaps in a side-by-side fashion and a preview of the created bitmap animation, would be very useful.

IV.B.3 Script Editing

Other than the ability to save created animation digraphs, there is no facility for the user to "record" his or her session with the interface. These recorded sessions, or *scripts*, are believed to have important applications in the learning, teaching and researching realms

[14]. The ability to save a "history of the user" session to a file and play it back would greatly enhance the value of the interface. The ability to edit, cut, paste and duplicate portions of scripts would complement the abilities to save and play back scripts

The AGE animation environment allows the user to save and playback the contents of a user-session [14]. This provision can be used as part of an editing facility.

IV.B.4 Unusual Machinery

Currently, the interface provides the ability to execute applications on Unix machines within the local network. However, the need for the interface to span across different machines other than Sun Sparc Stations is evident. A typical example of this is interaction with Silicon Graphics machines. Although the applications can be executed on them, they cannot display their output on non Silicon Graphics terminals. Silicon Graphics provides a distributed graphics library (dgl) that enables other machines to execute and compile programs written using dgl. But the display still has to be on a Silicon Graphics terminal. Perhaps with the advent of X Windows there will be a way to get around this problem. Execution of applications on other machines such as the Cray, MasPar, and N-cube also present similar problems due to their unusual architectures and "Nontraditional Unix" operating systems.

IV.B.5 Programming Language Constructs

The addition of programming language constructs to the animation digraphs would open the door for the interface to be used as a visual programming tool. Provisions such as self-loops, conditional execution, sequencing, and interaction with actual programming pseudo-code would make the animation digraph not just a map for the sequence of animation

execution, but also a visual representation of programming constructs.

REFERENCES

[1] J. Abello, S. Sudarsky, T. Veatch, and J. Waller, "AGE: An Animated Graph Environment," DIMACS Workshop on Computational Support for Discrete Mathematics, March 12-14, 1992, Rutgers University, New Brunswick, NJ.

[2] Naba Barkakati, *X Window System Programming*. SAMS, Carmel, IN, 1991

[3] Marc H. Brown, *Algorithm Animation*, MIT Press. Cambridge, MA, 1988

[4] *Explorer Environment User's Guide*, Silicon Graphics, Mountain View, CA, 1991

[5] Hopcroft, J. E. and Tarjan, R. E., "Efficient Planarity Testing." *Journal of the ACM*. **21** (1974).

[6] Mark A. Linton, John M Vlissides, and Paul R. Calder, "Composing User Interfaces With Interviews," *IEEE Computer*, pp. 8-22, February 1989

[7] Brad A. Myers, Dario A Guise, Roger B. Dannenberg, Brad Vander Zanden, David S. Kosbie, Edward Pervin, Andrew Mickish, Phillipe Marchel. "Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces," *IEEE Computer*, pp. 71-85 November 1990.

[8] Mark Overmars, *Forms, A C-Library for Dialogues*. Department of Computer Science. Ultrecht University, The Netherlands, 1991

[9] Frances Newbery Paulisch and Walter F Tichy. "EDGE: An Extendable Graph Editor," *Software Practices and Experiences*, vol 20. pp. 63-88. June 1990

[10] John T. Stasko, "Tango. A Framework and System for Algorithm Animation" *IEEE Computer*, pp. 71-85, September 1990.

[11] Sandra Sudarsky, "Primitives for Algorithm Animation," M.S. Thesis, Department of Computer Science, Texas A&M University, December 1991.

[12] *Sun C Programmer's Guide*, Sun Microsystems, Inc . Mountain View, Ca., February 1991.

[13] Ray Swartz, *Unix Applications Programming, Mastering the Shell*. SAMS, Carmel. IN. 1990.

[14] A.S. Tanenbaum and S.J. Mullender, "An Overview of the Amoeba Distributed Operating System," *Parallel Computers and Computations*, edited by J. van Leeuwen and J.K. Lenstra, Mathematisch Centrum. Amsterdam. 1985

[15] Timothy R. Veatch, "AGE: A Distributed Environment for Creating Interactive Animations of Graphs," M.S Thesis, Department of Computer Science, Texas A&M University, December 1990

APPENDIX A

USER MANUAL

A.A Interface Access

The user must be logged onto a computer that is operating under the Unix operating system. Once inside Unix, the windowing environment is placed into the X Window System. Within the X Windowing System, the current directory is changed to the directory where the interface executable resides. On our system the directory is */user/agesw/Interface/bin/Interface.* The Interface is invoked by calling the executable name "IntApp". After a few seconds of processing, the interface window appears and should look similar to figure 21.

A.B External View

The interface is laid out in a simple, easy to use fashion. It is broken into five visual components: the *Menu Bar*, the *Animation Display Area*, the *Control*, the *Canvas* and the *Error Box*. On the top of the interface is the Menu Bar. The Menu Bar enables the user to return to Unix and gain interface information dealing with interface usage and version. Directly beneath the Menu Bar is the Animation Display Area. The Animation Display Area is a reserved area in the interface window for the placement of algorithm animation environments. Figure 21 displays no animation environment. Beneath the Animation Display area are the Control and Canvas. The Control is a collection of buttons providing the user means to invoke and terminate processes, execute and stop animation digraphs, and

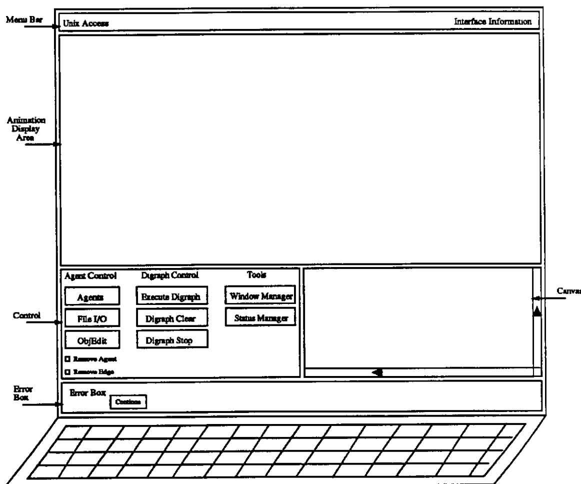


Fig. 21. Interface Appearance Upon Invocation

monitor network statistics. The Canvas, residing to the right of the Control, displays the current animation digraph. The user may reposition the icons composing the animation digraph and create edges between icons to represent the informational path flows and execution order. The lowest portion of the interface is the Error Box. Any problems, errors or unusual events occurring during a user session are reported to this subwindow, along with helpful suggestions

Fig. 22. Menu Bar

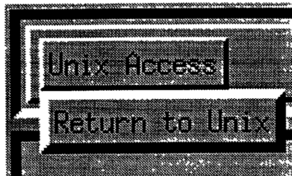


Fig. 23. Unix Access Menu

A.C Menu Bar

The top portion of the interface consists of the Menu Bar which spans the width of the screen. The words "*Unix Access*" reside on the leftmost end of the Menu Bar while "*Interface Information*" resides on the right, see Figure 22. By selecting the phrase "Unix Access" with the left mouse button, a push button with the phrase "*Return to Unix*" appears beneath, see Figure 23. By selecting "Return to Unix" with the left mouse button, a popup message appears querying the user's intentions. The user may select the "*Continue*" button to exit the system or "*Cancel*" button to return to normal operation.

If the users selects the word "Interface Information" with the left mouse button, two push buttons appear beneath, see Figure 24. The buttons are labeled "*Usage*" and "*Version*". By selecting the "Usage" button with the left mouse button, a popup window appears giving full interface instructions. The popup window is removed by selecting the "*Close*" button beneath the instructions. By selecting the "Version" button with the left mouse button, a

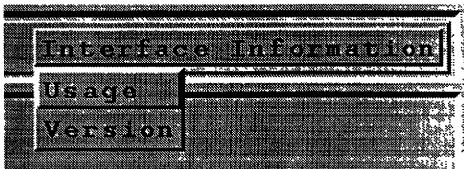


Fig. 24. Interface Information Menu

popup window appears displaying program name, version, creation date, author, and any other up to date information the user may need. Similar to the help window, the version window is removed by selecting the "Close" button.

If at any time the user decides not to select one of the optional buttons displayed from the menu bar, he is to click the menu bar with the left mouse button anywhere in between the two phrases on the Menu Bar. This activity removes the optional buttons. For Example, if the users is presented with the "Help" and "Version" buttons but wants neither, he simply clicks the Menu Bar with the left mouse button to remove the buttons.

A.D Animation Display Area

The area immediately beneath the Menu Bar is the Animation Display Area. This subwindow acts as a place holder for an algorithm animation environment when it is invoked. All user interaction with this area, other than with an animation environment, is ignored.

A.E Control

Beneath the Animation Display Area lies the Control and Canvas subwindows. The Control subwindow is left of the Canvas, see Figure 21, 25.

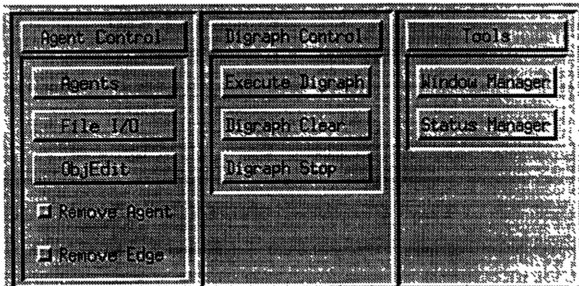


Fig. 25. The Control Subwindow

The Control is broken up into three separate subwindows. These subwindows are labeled "*Agent Control*", "*Digraph Control*" and "*Tools*". The "Agent Control" subwindow provides tools for loading individual processes or digraphs on to the interface and removing specific processes. The "Digraph Control" box provides tools for executing, stopping, or erasing the current animation digraph. The "Tools" box provides tools for monitoring the status of applications on the animation digraph and viewing graphics generated by the applications.

A.E.1 Agent Control

The "Agent Control" subwindow contains three buttons labeled "*Agents*", "*File I/O*", and "*ObjEdit*", and two toggle buttons labeled "*Remove Agent*" and "*Remove Edge*".

A.E.1.1 Agents

By selecting the "Agents" button the user will be given a popup menu with three buttons: "*Environment*", "*Client*", and "*Executable*", see Figure 26. The user must select

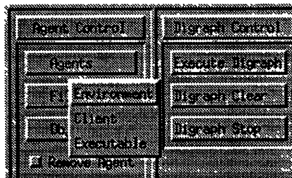


Fig. 26. The Agents Submenu

one of the three buttons with the right mouse button. If the user selects any portion of the screen except one of the three buttons, the popup menu will disappear.

Environment The "Environment" button allows the user to invoke an algorithm animation environment, such as AGE, Tango or Balsa, into the interface digraph. When the "Environment" button is pressed with the right mouse button, the user will be given a "SysLoad" popup window, see Figure 27 . The "SysLoad" window is broken into two main portions, the machine list section and the items list section. The leftmost section is the machine section. It contains a list of all available machines on the local network. A slider runs vertically next to the list of machines. By moving the slider up and down, the user may control which machines are visible in the selection window. The user must select a machine to execute the animation environment with the left mouse button.

The rightmost section of the "SysLoad" window contains a selection box with three buttons. Inside the box, labeled "Items" are the choices for algorithm animation systems. If there are more systems then there is room, then a slider will control the visibility of items. The user must select one of the options with the left mouse button. If a selection has been made, the choice will be displayed in a box beneath the "Items" box labeled "Selection". If

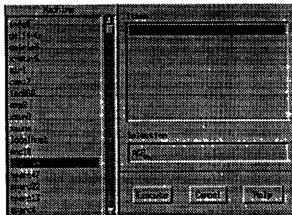


Fig. 27. The Sysload Popup Window

the user chooses, he may type in the selection into the box without actually selecting from the "Items" box.

Once a machine and the desired algorithm animation environment have been selected the user may invoke his choice by pressing the "Execute" button with the left mouse button. If the "Execute" button is pressed with out a selected machine, the interface will assign a default machine. If an algorithm animation system in not selected, the user will receive an error popup window that can be removed by pressing the "Close" button contained in the error window. If the user so chooses, he may also select the "Cancel" or the "Help" buttons. The "Cancel" button will return the user to the main interface. The "Help" button will will cause a popup window to appear that explains how the "SysLoad" window works. Once again this popup can be removed by pressing the "Close" button contained within it.

If the user types in a reply to the "Selection" box of the "SysLoad" window that is not not known by the interface, a popup window will appear querying the user if he would like to add the reply to the system. The user must choose either the "Cancel" or "Define" button. The "Cancel" button returns the user to the "SysLoad" window. The "Define" button creates a new interactive window, which queries information from the user, called

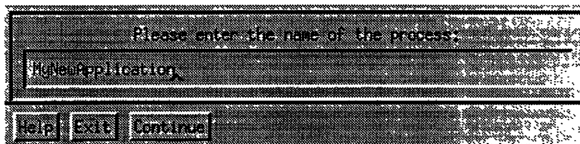


Fig. 28. Query Box

the "*Query Box*", see Figure 28.

Query Box The first item requested by the "Query Box" is the name of the application. The users is to type the name into the box provided. After entering the name, the user must press the "Continue" button. The user will then be asked for the entire path of the executable. The continue button proceeds. The user will then be asked for the number of stationary bitmaps. The stationary bitmaps are the bitmaps that cycle in the icon while the process is not executing. Once the number is entered, the path for each bitmap is requested. After the stationary bitmaps are entered, the user enters the same information concerning the executing bitmaps. The executing bitmaps are the bitmaps that are cycled in the icon while the process is executing. Once the information for the executing bitmaps has been received the user is asked to give the number of acceptable inputs. An exceptable input is a set of integers, floats, words, graphs, filenames, environment specific data and matrices the process needs to execute. A process may have more than one acceptable input. Once the number is suplied, the users is asked for the number of integers, floats, words, graphs, filenames, environment specific data and matrices that are needed for that particular input. Once all the input information has been given, the same type of questions will query the user for the format of the output. However, each process is allowed only one acceptable

output format. Following the output format, the application type will be queried. There are three type of application: Animation Environment (601), Animation Client(602), and general executable (603). If the application is an animation environment, then the user will be queried for the path of the I/O Master for that environment.

Client Choosing the "Client" button from the "Agents" menu will produce a window labeled "ClientLoad". This new window allows the user to invoke an algorithm client to operate with a selected algorithm animation environment. Since AGE is one of the environments this interface was designed for, all animations that execute on the AGE "server" are AGE "clients". The "ClientLoad" window operates with the exact same behavior as the "Sysload" window. The only difference is the user will be selecting animation clients rather than systems.

Executable Choosing the "Executable" button from the "Agents" menu will produce a window labeled "ProcLoad". This window allows the user to invoke any process to be placed inside the animation digraph that is not an algorithm animation environment or a client associated with a environment. Once again, the "ProcLoad" window behaves in the exact same fashion as the "SysLoad" and "ClientLoad" windows.

A.E.1.2 File I/O

The "File I/O" button in the "Agent Control" subwindow enables the user to save or load pre-existing animation digraphs or catalogs. A catalog is a list of available processes the user has to select from. The catalogs are stored as files in the user's directory. Upon pressing the "File I/O" button, a popup menu with two buttons, labeled "Digraph" and "Catalog", will appear. Next to each label is a small arrow. When the user places the cursor

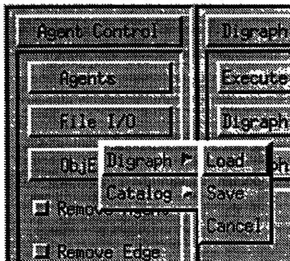


Fig. 29. File I/O Submenu

over either of the arrows, another popup menu will appear. The menu has three buttons : "Load", "Save" and "Cancel", see Figure 29. The menu from the "Digraph" arrow applies to the loading and saving of animation digraphs and the menu from the "Catalog" applies to catalogs. The user must make a selection with the right mouse button on any of the menu items. If the buttons is pressed on a non-menu item, the menus go away.

Load Pressing either of the "Load" buttons will invoke a loading popup window, see Figure 30. The window is broken into five major sections: the "Filter" window, the "Directories" and "Files" lists, the "Selection" window and the control buttons. The user may control which directory the file is loaded from by manipulating the "Directories" list. By double clicking with the left mouse buttons on any of the directory options, the list will descend into that directory. Similarly, the "Files" list displays which files are available in the chosen directory. The user needs to only click the file choice once with the left mouse button to make a selection. The selection will be displayed in the "Selection"' window. The full path is displayed with the name. The user may wish to type in his own entry into the

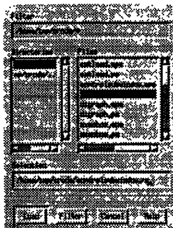


Fig. 30. File Load Window

selection window instead of choosing from the list. There is an alternative method of making a file selection. The user may wish to filter out certain files from the directory. He does this by modifying the filter in the "Filter" window. For example, if the user wanted only ".NTK" files displayed in his "Files" list, he would enter the entire path into the "Filter" window and end the entry with "*.NTK". In this example, the "*" is a wild card. Pressing the "Filter" Button will engage the filter and only ".NTK" files will be observed in the "Files" list. The "Load" button loads the selected file into the interface. The "Cancel" button returns the user to the interface and the "Help" button displays a helping popup window.

Save Pressing either of the "Save" buttons invokes a saving popup window very similar to the loading window described above, see Figure 30. All sub windows behave the same as their loading counterparts, except the user must type in a new name to the "Selection" window. If a new name is not supplied, the contents will overwrite the file selected in the "Files" and "Selection" windows. The save button is the only different item and pressing that button executes the saving process on the selection.

A.E.1.3 ObjEdit

When the user selects the "ObjEdit" button from the "Agent Control" subwindow the user is given the opportunity to modify any of the attributes associated with a particular application. The users is given the exact same "Query Box" as if they were defining a process to system for the first time, see Figure 28. When the user supplies the name, the interface will search for its entry. When found, the old values will be used as the default values for the queries. Once again if a new name is given, the user will be given the opportunity to define it.

A.E.1.4 Remove Agents

The "Remove Agents" toggle button in the "Agent Control" subwindow places the user into kill mode where he may remove application processes from the animation digraph. If the cursor is placed over an existing application icon in the animation digraph and the left mouse button is pushed, that application is terminated. As long as the "Remove Agents" button is engaged, the user can remove applications from the animation digraph.

A.E.1.5 Remove Edges

The "Remove Edges" toggle button in the "Agent Control" subwindow places the user into edge kill mode. While the button is engaged, the user may remove edges. Edges are removed the same way edges are created. The initial icon is pressed with the middle button. The user then presses the receiving icon with the middle button. If there exists an edge, it is removed.

A.E.2 Digraph Control

The "Digraph Control" subwindow contains three buttons labeled "*Execute Digraph*", "*Digraph Clear*" and "*Digraph Stop*". This subwindow gives the user control over the overall status of the animation digraph.

A.E.2.1 Execute Digraph

The animation digraph is executed by pressing the "Execute Digraph" button. To execute the animation digraph, an initial process application must be chosen first. This is done by pressing the selected icon with the left mouse button. The chosen process is said to be "*Armed*" and is identified by a yellow frame around the icon in the animation digraph. The "Armed" icon acts as the initial starting point for digraph execution. The animation digraph begins execution at the "Active" application upon the pressing of the "Execute Digraph" button. An executing application is represented by a green frame around its corresponding icon and an animating bitmap image inside the icon. How ever, if the application process is instructed to execute and does not have the needed input, it's icon frame will turn orange. If the "Execute Digraph" button is pressed while there is no "Active" process, nothing happens.

A.E.2.2 Digraph Clear

The "Digraph Clear" button allows the user to remove all of the application processes from the animation Digraph in one command. Since the current animation digraph is not replaced when a new digraph is loaded in, the old animation digraph will have to be removed if a clean Canvas is desired.

Fig. 31. Interface Status Manager

A.E.2.3 Digraph Stop

The "Digraph Stop" button stops an executing animation digraph. The currently executing processes are stopped and the execution is no longer transferred. A stopped animation digraph is identified by blue edges on all icons. The animation digraph resumes execution with the pressing of the "Execute Digraph" button. Depending on the nature of the animation digraph and its processes, the digraph may need to be loaded in again to execute correctly. For example, if the animation digraph contains processes that do not contain self loops, then those processes will disappear after their execution. If the animation digraph is dependent on the output supplied by those processes, then the digraph will need to load the data in again in order to execute once more.

A.E.3 Tools

The "Tools" subwindow contains two buttons labeled "Status Manager" and "Window Manager". These two buttons are invocation buttons for tools that help watch the status of individual processes and the graphics each generates.

A.E.3.1 Status Manager

The "Status Manager" brings up a popup window that displays each process in the network as a row of information, see Figure 31. The information is surrounded by a frame the same color as the frame surrounding the process's icon. Inside each status row is the number of each process in the network, the name of the executable, the machine it is executing on, its current status, all the the other processes it is connected to, the port which it receives connection requests from and the child's process id which the process is executing on. All processes of a session are displayed here even after they are killed since all processes are remembered in a table. The status field of the row reports all changes to the process as soon as they happen. At the top of the "Status Manager" window is a button labeled "Turn Off". This removes the status manager but does not destroy its current contents. The status manager is primarily a devise for monitoring the entries inside the global process table.

A.E.3.2 Window Manager

The "Window Manager" is currently not incorporated thus its button has been desensitized.

A.F Canvas

The Canvas is used to display the interface animation digraph. It is the rectangular area to the right of the Control subwindow. The animation digraph is constructed here and during execution, some control of digraph execution may be exerted. The Canvas contains a virtual work area larger than the provided viewport thus vertical and horizontal scrollbars

TABLE I

Icon Color Status

Color	*Status*	*Meaning*
Red	Stopped	Application is waiting to execute or has already executed.
Yellow	Armed	Application is ready to execute and is waiting for execution signal.
Orange	Active	Application has received execution signal but is waiting for input data.
Green	Executing	Application has sufficient input to execute and is currently executing.
Blue	Halted	Animation digraph received stop signal during execution.

are supplied to control the view area. The animation digraph consists of nodes and edges where the nodes are displayed on the Canvas as icons and the edges are displayed as arrows.

A.F.1 Icons

Icons are the visual representation of the application processes in the animation digraph. Icons consist of a multi-colored frame, a pictorial bitmap and text title. Icons are movable and may be placed anywhere within the Canvas work area. They have limited control on an executing animation digraph. They are also used to designate the starting application for an animation digraph. When an application process terminates, its icon is removed form the Canvas.

Icon Frames The frame surrounding each icon is composed of two colors. The upper left corner is the unique color assigned to the icon from the interface. The color allocation is random and no two colors on the same animation digraph are exactly the same. The lower right corner reflects the application's current status. The color statuses are represented in

Table I.

Icon Bitmaps The image of each icon is an animated pictorial bitmap. The bitmap may have two animation states: non-executing and executing. In each state, the image is composed of multiple bitmaps which are displayed in flipbook cycle fashion. The icon bitmaps are "changed" once every second. If no animation is desired, then only one bitmap for that state is supplied. For each state, the number of bitmaps is limited by the ability of the computer the interface is executing on.

Icon Control Icons are fully movable, let the user stop and start the animation digraph, and allow the user to select a starting application for the animation digraph. The icon also has the ability to terminate the application process it is representing.

Icon Movement Icons are moved by pressing the icon with the left mouse button, dragging the icon to its new position, and releasing the mouse button. An icon may be positioned anywhere on the Canvas work area. Any edges associated with the icon will move along it.

Icon Digraph Manipulation The icon gives the user the ability to stop and restart an executing animation digraph. It also gives the user the ability to terminate its application process. When the icon is pressed with the right mouse button, a menu appears with three buttons: "Activate", "Stop" and "Kill", see Figure 32.

The "Activate" button allows the user to restart a "Halted" animation digraph, the "Stop" button halts an executing animation digraph and the "Kill" button kills the corresponding application process.

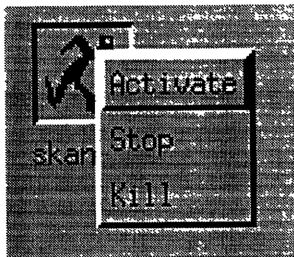


Fig. 32. Animation Digraph Icon Menu

Animation Digraph Initiation If an Icon is not "Active", "Executing" or "Stopped", or none of the other icons in its local animation digraph are "Active" or "Executing", it is placed in the "Armed" status when the user presses the icon with the left mouse button or moves it. This is the interfaces' method for designating a starting node for the animation digraph. Before any edges connect icons, all can be made "Armed" and thus executed at the same time. However, only one Icon within a local animation digraph is allowed to be "Armed".

A.F.2 Edges

Edges connect the icons in an animation digraph. They represent the directional flows of data and the application execution sequence. They are seen visually as arrows between two icons.

Edge Manipulation Edges are created by clicking the initial icon with the middle mouse button. An arrow attached to that icon will follow the cursor. By clicking a second icon with

the middle mouse mouse, the edge is complete and connects the two icons. If something other than an icon was pressed once an edge has started, the edge is terminated. Edges may be removed by pressing the "Remove Edge" toggle button in the Control subwindow. Once the toggle is pressed, the interface removes edges between icons instead of creating them. The user clicks an initial icon with the middle mouse button. When a second icon is clicked with the middle mouse button, the edge connecting the two icons is removed. Nothing occurs if a deleting edge is drawn between two unconnected icons.

If two icons are not in the same local animation digraph when the user connects them, the interface will cause the second icon to enter the "Stopped" state as soon as an edge is created. This ensures only one starting application within a local animation digraph.

A.G Error Box

The Error Box receives error messages for the interface. As it receives a message, the message is displayed and a button labeled "Continue" is created. The message stays in the window until the "Continue" button is pressed. The "Continue" button's purpose is to remove the error message. Leaving the error message in the window does not affect the behavior of the interface.

APPENDIX B

APPLICATION REGISTRATION

Although not a large part of the external view of the interface, the application administration plays a vital portion in the expandability of the interface. The registration of new applications is not a difficult procedure but one that must done with thought and care. Simple errors will not only have direct effects on how an application behaves but how it interacts with other processes as well.

B.A Needed Information

For every application that executes on the interface, there is a set number of informative data elements describing the application and communicative behavior. Data concerning the name and executable are needed for the Communication Agent. Bitmap information is needed to display the appearance of the icon in the Canvas. Input and output sequences are needed to inform the interface how it communicates with other processes.

For the Communication Agent to launch an application properly, the application's name and executable path are required. If an Agent tries to launch an application that does not exist at the given path, an error will be displayed in the Error Box and the process will be removed from the object table. The interface will continue making the error if not corrected. The type of the application is also needed. The interface understands three different types of applications: Animation Environments, Clients and general executables. Animation Environments have a type number of 601 and need executable paths for IO

Masters that are to communicate between them and the interface. Animation Environments are executed immediately after being launched from the Communication Agent. The IO Master is launched as soon the first client is launched. Clients are considered applications who need an Animation Environment to operate. Clients have a type number of 602 and are prevented from launching until an Animation Environment is provided. General executables refer to any applications that are independent of environments or clients. Executables have a type number of 603 and have no constraints on launching.

B.B Catalogs

All information defining an application, except for the input and output sequences, is stored in files called *Catalogs*. The Interface uses a default catalog stored in *.XMIProcTable* which is loaded during startup time. Each catalog contains the necessary information for many applications. The interface only allows one catalog to be in the interface at a time but the user can control the work environments by controlling the catalogs used.

Each entry in a catalog file contains the following:

- Application Name
- Application Executable path
- Number of Non-Executing Bitmaps
- Path for each Bitmap
- Number of Executing Bitmaps
- Path for each Bitmap
- The path for the header file containing I/O Information
- Application type

- Path of IO Master if Animation Environment

The application name is the name used in the icon representation in the animation digraph. The executable path tells the Communication Agent where to find the application executable. The number of non-executing bitmaps is the number of bitmaps that make up the animated icon cycle when the application is not executing. For each of the number of bitmap paths, there must be the path of where each bitmap can be found. The number of executing bitmaps and their paths are defined similarly. The path for the header file tells the interface where to find the header file for an application. The type of application tells the interface whether it is an animation environment (601), animation environment client (602), or general executable (603). If the application is an algorithm animation environment, the interface then needs the path of the I/O Master for that environment so it can be executed with the clients.

The first element of a catalog file contains the number of entries within that file. The individual entries follow. Here is an example of the contents of a Catalog file containing three applications, *Filewriter (executable)*, *AGE (environment)*, and *Connectivity (client)*:

```
3


FileWriter

/user/croda/Research/X/New/IOdo/FileWriter

1

/user/croda/Research/X/New/Bitmaps/filwrite1

2

/user/croda/Research/X/New/Bitmaps/filwrite1
```

```
/user/croda/Research/X/New/Bitmaps/filwrite2
/user/croda/Research/X/New/Headers/FileWriter.hdr
603


AGE
/user/agesw/NEWAGE/pub/bin/AGE
1
/user/croda/Research/X/New/Bitmaps/AGE.1
3
/user/croda/Research/X/New/Bitmaps/AGE.1
/user/croda/Research/X/New/Bitmaps/AGE.2
/user/croda/Research/X/New/Bitmaps/AGE.3
/user/croda/Research/X/New/Headers/AGE.hdr
601
/user/croda/Research/X/New/IOMaster/IOMaster


Connectivity
/tmp_mnt/cssun/xy1b/agesw/NEWAGE/clients/connectivity/connected
1
/user/croda/Research/X/New/Bitmaps/connected.1
5
/user/croda/Research/X/New/Bitmaps/connected.1
/user/croda/Research/X/New/Bitmaps/connected.2
```

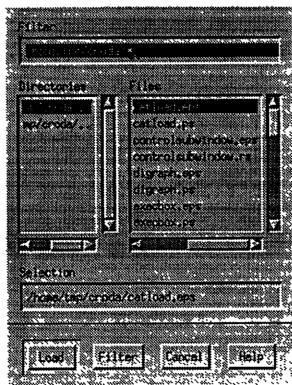


Fig. 33. Catalog Loading Menu

/user/croda/Research/X/New/Bitmaps/connected.3

/user/croda/Research/X/New/Bitmaps/connected.4

/user/croda/Research/X/New/Bitmaps/connected.5

/user/croda/Research/X/New/Headers/connectivity.hdr

602

The user may load catalogs by using the *Catalog Loader* which is accessed via the Control button "File I/O". After the button is pressed, a menu appears giving the options of "Catalog" or "Digraph". These buttons cascade to further options of "Load", "Save" or "Cancel". The button sequence "Catalog → Load" produces a loading menu called *CatLoad*. See figure 33. This tool enables the user to select a catalog and load it into the interface.

TABLE II

Format String Decomposition

String Position	*Data Type*
0	Integer Values
1	Float Point Values
2	Words (text strings)
3	Graphs
4	Filenames (text strings)
5	Environment Specific Data
6	Matrices

B.C Header Files

Each application registered with the interface has a header file which instructs the interface how to conduct the communication for that application. Each header file contains a number of sequences that indicate the format of input the application expects and the format of its output. Each format consists of an string of ten integers. Each position of the string indicates the number of data elements associated with that sequence. The significance of each position is is displayed in Table II:

There are three abstract data types in each sequence: Graphs, Environment Specific Data, and Matrices.

Graphs The format of Graphs is as follows:

- Number of vertices in Graph (integer).
- Number of edges in Graph (integer).
- For each edge, a pair of integers identifying the edges two vertices.

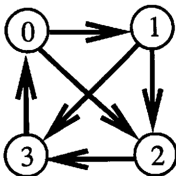


Fig. 34. Sample Graph

The interface assumes the edges are directed. The vertices given for each edges must be between or including 0 through the number of vertices minus one. An example of the graph representation for Figure 34 is:

```
4
6
0 1
1 2
2 3
3 0
0 2
1 3
```

Environment Specific Data The format of environment specific data is a semaphore. Since this type of data refers to a type of data that can't be universally translated, we use the animation environment as the communication medium. For example, in the AGE environment, one client may output an AGE GRAPH to another client. Since AGE GRAPHS are environment specific, the first client simply outputs an integer and when that integer

arrives at at the other client, it knows it can read the data from the AGE window.

Matrix The format of a Matrix is as follows:

- Number of rows in Matrix (integer).
- Number of columns in Matrix (integer).
- Each of the rows X columns entries (integers).

An example of a Matrix is the adjacency matrix for the Graph in Figure 34:

4

4

0 1 1 0

0 0 1 1

0 0 0 1

1 0 0 0

Header File Contents The header files contains the I/O sequences for the application. Their format is defined as follows:

1) #INPUT
2) Number of Necessary Input Sequences (at least one)
3) An integer string of length 10 where each digit describes the number of needed: Integers, Floats, Words, Graphs, File Names, Environment Data, and Matrices
4) #OUTPUT
5) Number of Output Sequences (always one)

6) An integer string of length 10 where each digit describes the number of output: Integers, Floats, Words, Graphs, File Names, Environment Data, and Matrices

An example header file would look like the one for connectivity.hdr:

```
#INPUT
1
0000010000
#OUTPUT
1
0000010000
```

From this file example, the Connectivity client can receive only one AGE GRAPH as input. It outputs one AGE GRAPH.

B.D Application Registration

If the user chooses, he may create his own catalog file. By using one of the available UNIX file editors (vi, emacs) the user may design his own catalog that should look very similar to the given example. In the same manner, the user may create a header file for an application.

The user may also define a new application into a catalog by pressing the "ObjEdit" button in the Control subwindow. Upon pressing the button, a window appears called the *Query Box* which prompts the user for information then saves the results to the current catalog. See Figure 35. The questions in the Query Box are self explanatory and the user

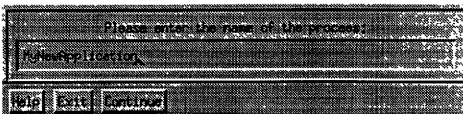


Fig. 35. Query Box

always has the option of exiting the procedure. The questions asked are sufficient to add a new application to an already existing catalog and create a new header file.

B.E Application Input

If an application wishes to receive input, it must do so through standard input. The application's Communication Agent will send it the data when the application is allowed to execute. The application must read all data elements in the order of the described sequences. Likewise, all abstract data will be sent in their format described previously. For example, if an application wishes to read in the input sequence:

3000011000

the corresponding C code would look like:

```
fscanf(stdin, "%d", &integer1);
fscanf(stdin, "%d", &integer2);
fscanf(stdin, "%d", &integer3);


fscanf(stdin, "%d", &envsemaphore);


fscanf(stdin, "%d", &rows);
```

```
fscanf(stdin, "%d", &columns);

for(i = 0; i < rows; i ++)

   for(j = 0; j < columns, j++)

      fscanf(stdin, "%d", &matrix[i][j]);
```

B.F Application Output

If an application wishes to transmit data, it must do so through standard output. The application's Communication Agent will receive the data when the application transmits it. Each data element must be followed by a "\n" in order to separate individual elements. The application must transmit all data elements in the order of the described sequences. Likewise, all abstract data must be sent in their format described above. Once the data has outputted all the data, it must perform the system call "fflush(stdout)". This command will flush the standard output channel so the Communication Agent can read it. For example, if the application wishes to output the data sequence:

0201100000

the corresponding C code would look like:

```
fprintf(stdout, "%f\n", float1);

fprintf(stdout, "%f\n", float2);


fprintf(stdout, "%d\n", numvertices);

fprintf(stdout, "%d\n", numedges);

fprintf(stdout, "%d\n %d\n", vertex0, vertex1);

fprintf(stdout, "%d\n %d\n", vertex1, vertex2);
```

```
fprintf(stdout, "%d\n %d\n", vertex2, vertex0);


fprintf(stdout, "%s\n", filename1);

fflush(stdout);
```

B.G Application Self Loops

If the user wishes the application to stay in the animation digraph after it has been completed once, he or she must place it in some kind of loop. If the loop is not available, the next time the application executes it will terminate. Only the looping behavior of an application will prevent it from leaving the animation digraph before the user wishes.

APPENDIX C

I/O MASTER CREATION

The I/O Master is composed of three computational mechanisms: the *Command Router*, the *Environment Liaison* and the *Output Dispatch*. The Command Router receives the interface commands along with the client information and the needed client data. The notification of special events are sent back to the interface. The Command Router sends the needed client data to the Environment Liaison and gets back environmental status information. It sends the client information to the Output Dispatch. The Environment Liaison sends requests to the animation environment and gets back user suplied data and special device events. The received data is relayed to the Output Dispatch where it is sent to the Communication Agent of the client needing input data.

C.A Command Router

The *Command Router* receives all communications from the interface and distributes the information to the Environment Liaison or the Output Dispatch. Its purpose is to communicate with the main interface and route the received commands to the proper destinations. Like the Communication Agent Input Reception, the Command Router blocks the I/O Master until a message arrives from the interface. The message is immediately interpreted and sent to its proper channels.

Most of the code for the Command Router has been supplied in the file "CommandInterpreter.c". This file acts as the "main" for the I/O Master. It creates connections to the

TABLE III

Major Functions of CommandInterpreter.c

Function	*Description*
main	Hook up to Animation Environment and Interface, enter into polling loop.
initCIC	Become an internet socket client to the interface.
initIclient	Socket connect routine for initCIC.
enterInterfacePoll	Enter I/O Master into polling loop.
interfacePoll	Polls interface socket and Environment sockets for incoming messages.
setTimer	Sets the timer used for the *select* command.
setFds	Tell the *select* command to poll the interface and animation environment's sockets.
handleInterfaceCommands	Process messages from interface.
handleClientConnect	Hook up to Communication Agent of environment client and get data from the animation environment.
getNeededInput	Tell the *Environment Liaison* to get data and relay it to the *Output Dispatch.*

algorithm animation environment and the interface then goes into a polling loop where it check for messages from the interface and the animation environment. Once a command comes through it relays the message to the *Environment Liaison* or the *Output Dispatch.* Return controls to the polling loop were the I/O Master stays until needed again.

The break down of the major functions in CommandInterpreter.c are given in Table III. When the reader is creating a new I/O Master he or she will only have to change those parts involving the *Environment Liaison.*

C.B Environment Liaison

The purpose of the *Environment Liaison* is to act as a client to the desired animation environment. The needed client data is translated into requests and sent to the environment. The results of the requests are sent back to the Liaison and relayed to the Output Dispatch.

TABLE IV

Major Functions of AGELiaison.c

Function	*Description*
AGEConnect	Connects the I/O Master to the AGE environment.
AGEEvent	Receives a message from AGE and reports it to *Command Router.*
getAGEInt	Get integer data from AGE and send to Output Dispatch.
getAGEFloat	Get floating point data from AGE and send to Output Dispatch.
getAGEWord	Get word data from AGE and send to Output Dispatch.
getAGEGraph	Get Graph data from AGE and send to Output Dispatch.
getAGEFilename	Get file name from AGE and send to Output Dispatch.
getAGESys	Get system data from AGE and send to Output Dispatch.
getAGEMatrix	Get Matrix data from AGE and send to Output Dispatch.

The Environment Liaison communicates with the animation environment according to the environment's methods.

The Reader creating a new I/O master will have to create most of this code. An example from the AGE environment is given in "AGELiaison.c". The connection instructions to the AGE animation environment are included. The code also contains instructions how to get each type of data from the animation environment and how to send it to the *Output Dispatch.* The breakdown of the major functions of "AGELiaison.c" are given in Table IV.

C.C Output Dispatch

The purpose of the *Output Dispatch* is to connect with the Communication Agent of the client needing input and transmit data to it once the information arrives from the Environment Liaison. Unlike the Output Dispatch of the Communication Agent, information is sent to only one connected Communication Agent at a time. If the I/O Master retrieves information for the same client at a later time, it does not need to reconnect with that

TABLE V

Major Functions of OutputDispatch.c

Function	*Description*
clientNotConnected	Returns true if an input client is not connected.
newClient	Creates a connection to a Communication Agent.
addClient	Add a new client connection to an internal list.
sendMessageData	Takes the data from the Environment Liaison and transmits it to the client's Communication Agent.

client's Communication Agent.

Most of the code needed for the Output Dispatch is already supplied in the file "Output-Dispatch.c". In fact, because its routines are called by both the *Command Router* and the *Environment Liaison* and calls neither, no code should have to be rewritten. The Output Dispatch keeps a list of all the clients it has attached to so it does not attach to the same one twice. Otherwise, it receives commands to connect from the Command Router and receives data to transmit from the Environment Liaison. The breakdown of the major functions of "OutputDispatch.c" are given in table V.

APPENDIX D

GLOSSARY

Agent Information Data group of Object Table containing execution information about each Communication Agent and its child applications.

Animation Digraph Basic interface input consisting of a directed graph.

Animation Display Area Mini handler in User Event Handler which controls events in the Animation Display Area of the interface which reserves an area for the display of an algorithm animation environment.

AGE Animated Graph Environment. Algorithm animation environment created at Texas A&M University by Abello, Sudarsky, Waller and Veatch [1].

Balsa One of the first algorithm animation environments created at Brown University by Marc Brown [3].

Bitmap Rectangular array of pixels, where each location contains an On/Off state for that pixel [2].

Biconnected Component A subgraph that does not contain any vertex whose removal will disconnect the graph.

Bulletin Board Internal mechanism of Communication Agent that stores received information and judges executable condition of child application.

Canvas Mini handler in User Event Handler which controls the events in the Canvas subwindow which displays the interface animation digraph.

Catalog File used by interface to store essential information pertaining to each application registered with the interface.

Catalog Loader interface tool used to query user for desired catalog to be used in interface.

Command Router Internal mechanism of Communication Agent which controls its decision making.

Command Router(I/O Master) Internal mechanism of I/O Master which receives all communications and distributes them among the other I/O Master internal mechanisms.

Communication Agent Fundamental element of interface connecting the interface and an application process.

Communication Digraph Internal software representation of interprocess communication network described by interface animation digraph.

Control Mini handler in User Event Handler which controls the events in the Control subwindow which manipulates the state and status of animation digraph.

Connected Component A subgraph such that for each pair of vertices, v and w, within the subgraph, there exists a path from v to w.

Edge List Data group of Object Table containing edge information of the animation digraph.

Environment Liaison Internal mechanism of I/O Master which acts as a client to the desired algorithm animation environment.

Error Box Mini handler in User Event Handler which receives and displays the error messages from other internal interface elements.

Exec Overlays the calling process with the named file, then transfers to the entry point of the core image of the file [12].

Fork Creates a new process. The new process (child process) is an exact copy of the calling process [12].

Header Information Data group of Object Table containing all input and output sequences for each interface application.

Input Reception Internal mechanism of Communication Agent which monitors all lines of communication to the Communication Agent.

Interface Animation Digraph Basic interface input consisting of a directed graph.

Interface Communication Communication engine for interface with Communication Agents and I/O Masters.

I/O Master Fundamental element of interface linking the algorithm animation environment with the interface.

LAD Laboratory for Algorithms Design. Computer laboratory where this interface, AGE and many AGE animations were created.

Launching Procedure Subgroup of Object Table that forks and execs all Communication Agents.

Menu Bar Mini handler in User Event Handler which controls the events in the Menu Bar subwindow which enables the user to return to Unix and supplies helpful interface information.

Moving Object Data group of Object Table containing visual information concerning the icon representation of applications in the animation digraph.

Object Table Internal global data base for interface.

Output Dispatch Internal mechanism of Communication Agent to relay information to all other Communication Agents.

Output Dispatch(I/O Master) Internal mechanism of I/O Master to send received data to executing algorithm animation.

Planarity An algorithm to test if a graph can be imbedded on a plane such that no two edges of the graph intersect [5].

Pipe A facility of Unix that provides a one way flow of data usually from one process to another on the same machine.

Query Box interface tool used to query user for information concerning executing information for an interface application.

Scripts A recording of a user session with an algorithm animation environment.

Sockets A facility of Unix that provide two way flows of data, across the internet, usually between two processes.

Stdin Default input process data channel.

Stdout Default output process data channel.

Tango One of the first algorithm animation environments created at Brown University by John Stasko [10].

User Event Handler Receives visual events from X Window Manager and distributes them to the Object Table and interface Communication.

VITA

Christopher Roda grew up on the East side of Cleveland Ohio. He received his Bachelor of Science Degree in Computer Information Science in 1989 from the Ohio State University in Columbus Ohio. In continuation of his education, he finished his Master of Science in Computer Science in 1992 from Texas A&M University in College Station Texas. He plans to pursue his fortune in the area of computer animation. Christopher Roda can be reached at his email address croda@cs.tamu.edu. His permanent address is 35985 Timber Ridge, Willoughby Ohio, 44094.