②

RL-TM-93-6
In-House Report
December 1993

DEVELOPING THE MULTIMEDIA USER INTERFACE COMPONENT (MUSIC) FOR THE ICARUS PRESENTATION SYSTEM (IPS)

Ingrid Bartnik

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

RL-TM-93-6 has been reviewed and is approved for publication.

APPROVED: [illegible]
JOSEPH CAMERA, Chief
Signal Intelligence Division
Intelligence & Reconnaissance Directorate

FOR THE COMMANDER: Garry W. Barringer
GARRY W. BARRINGER
Technical Director
Intelligence & Reconnaissance Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE December 1993	3. REPORT TYPE AND DATES COVERED In-House Jun - Aug 93

4. TITLE AND SUBTITLE DEVELOPING THE MULTIMEDIA USER INTERFACE COMPONENT (MUSIC) FOR THE ICARUS PRESENTATION SYSTEM (IPS)	5. FUNDING NUMBERS PE - 62702F PR - 4594 TA - 15 WU - J9
6. AUTHOR(S) Ingrid Bartnik	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Rome Laboratory (IRAE) 32 Hangar Road Griffiss AFB NY 13441-4514	8. PERFORMING ORGANIZATION REPORT NUMBER RL-TM-93-6
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (IRAE) 32 Hangar Road Griffiss AFB NY 13441-4514	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Alex F. Sisti/IRAE (315) 330-4518

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Rome Lab's Intelligence Technology branch provides modeling and simulation support to a variety of intelligence consumers and analysts, across a wide spectrum of application areas. Much of the research, development and capabilities demonstrations take place in the ICARUS Prototype Development and Demonstration Facility, named after the character from Greek Mythology who built the first prototype to solve a problem. In order to provide a cohesive development and demonstration architecture, as well as to advance the state-of-the-art in user interfaces and presentation techniques, the ICARUS Presentation System was envisioned, being comprised of two components: the Interactive VuGraph (InterVu) System and the Multimedia User Interface Component (MUSIC). This report documents the initial research, design and implementation of a prototype of the MUSIC system.

DTIC QUALITY INSPECTED 2

14. SUBJECT TERMS User interface Rapid Prototyping X Toolkit Multimedia X windows	15. NUMBER OF PAGES 24
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR

Abstract Rome Lab's Intelligence Technology branch provides modeling and simulation support to a variety of intelligence consumers and analysts, across a wide spectrum of application areas. Much of the research, development and capabilities demonstrations take place in the ICARUS Prototype Development and Demonstration Facility, named after the character from Greek mythology who built the first prototype to solve a problem. In order to provide a cohesive development and demonstration architecture, as well as to advance the state-of-the-art in user interfaces and presentation techniques, the ICARUS Presentation System was envisioned, being comprised of two components: the Interactive VuGraph (InterVu) System and the Multimedia User Interface Component (MUSIC). This report documents the initial research, design and implementation of a prototype of the MUSIC system.

1.0 INTRODUCTION

Rome Lab's Intelligence Technology branch has long provided modeling and simulation support to a variety of intelligence consumers and analysts, across a wide spectrum of application areas. While most of this support is in the form of specific, product-oriented models/simulations for specific users' stated problems (and usually contracted out), a good deal of the basic research is done in-house. To that end, the ICARUS Prototype Development and Demonstration Facility was built, for the purpose of advancing the state-of-the-art in modeling and simulation science.

1.1 The ICARUS Prototype Development and Demonstration Facility

The ICARUS Prototype Development and Demonstration Facility is an in-house computing environment in which R&D work is performed in modeling and simulation technology (e.g., object-oriented simulation, software/model reuse, model integration, metamodeling, etc.), while also serving as a facility for demonstrating to visitors these new techniques and some of the products built for operational customers. In order to provide a cohesive development and demonstration architecture, as well as to advance the state-of-the-art in user interfaces and presentation techniques, the ICARUS Presentation System was envisioned, being comprised of two components: the Interactive VuGraph (InterVu) System and the Multimedia User Interface Component (MUSIC).

1.2 The ICARUS Presentation System (IPS)

The ICARUS Presentation System (IPS) is a presentation layer which will be built on X Windows, and will serve two purposes. First, it will serve as a graphical user interface to all software systems hosted in the facility, and second, it will be used as an interactive briefing tool. These components are briefly described below.

1.3 Interactive VuGraph (InterVu) System

The Interactive VuGraph (InterVu) System is a new presentation technique which will support both the canned ICARUS briefing and special-purpose presentations that are interactively "guided" by members of the audience. InterVu, which will be a user-selectable option, will consist of a series of presentation screens (much like the present ICARUS storyboards), whose major points -- bullets and/or windows -- will be context-sensitive, allowing a briefing to be tailored to the specific interests of any audience. The InterVu System will also allow for demonstrations of any software system of interest to be launched on any of the ICARUS hardware; again, as interactively requested by the audience.

1.4 Multimedia User Interface Component (MUSIC)

The Multimedia User Interface Component (MUSIC) is designed to be the method of entrance to all software components hosted in the ICARUS facility. It will be a graphical, iconic interface that gives an ICARUS user virtual access to any software system, using a common "look and feel"; regardless of where the system is physically hosted on the ICARUS network. Initially, it will be based on a "point-and-click" methodology, but long-range plans are to incorporate other Rome Lab prototypes in voice recognition and Virtual Reality (VR) devices for input methods. It is this MUSIC component which is the subject of this report. Specifically, the following sections discuss the research, design and implementation of the initial prototype of the Multimedia User Interface Component (MUSIC) for the ICARUS Presentation System (IPS).

2.0 DESIGN CONSIDERATIONS

A significant step in developing the Multimedia User Interface Component (MUSIC) for the ICARUS Presentation System (IPS) was analyzing many design considerations, including the graphical user interface (GUI), platforms, and window systems. These are vital for designing the MUSIC and InterVu systems as shells that will have considerable dual use and commercial potential.

2.1 Graphical User Interface

A graphical user interface represents a distinctively different human computer interface (HCI) than the "traditional textual line interface" [3]. GUIs ordinarily provide the user with a graphical interactive method of directly manipulating display objects through various inputs. There are several advantages associated with GUIs.

By directly manipulating and interacting with display objects, the user has virtual and transparent access to the various applications the GUI supports. These objects include windows, icons, and popup/pulldown menus. Through these, the users do not have to be concerned with the internal operations of the various platforms or software systems. Instead, they merely need to know how to operate the GUI.

Through GUIs, modeless interfaces can be implemented. Modeless interfaces provide users with opportunities to access various states of the program, not just the current state that they are in. In modal interfaces, a user can only maneuver within the current state of the program. In order to access any other dimensions of the program, the user has to find the desired mode by going through a hierarchy of windows. This can become particularly disturbing when two sections of an application depend upon each other. Through display objects, users can simultaneously access the multiple modes within a program by dynamically transferring control through callbacks. One potential problem is creating an excessive amount of modes that ultimately confuse users and developers alike.

GUIs in turn tend to have a common look-and-feel. This is conveyed through various widget sets which are not the same, but contain significant similarities. (A widget is a user interface component). Each widget set allows at least the manipulation of, as mentioned, icons, windows, and menus such as pulldown and popup. Thus, the widgets of each set are similar in appearance as well as in function. The user can be provided with numerous interfaces that transparently access applications while not mandating that the user be familiar with interfaces in general, or with each individual application's interface.

2.2 Platforms

With the advent of multiple platforms, compatibility has become a major issue. In particular, user interfaces have to be portable to all hardware and not just customized for one specific system.

2.3 Window Systems

Another aspect that had to be acknowledged was window systems. Window systems "provide the underlying window graphics libraries and device drivers for the construction of window or graphical user interfaces" [3]. Currently there is a variety of window systems available. Some of those include X Window System, the Macintosh Toolbox, Microsoft Windows, and NeXTStep. In developing the IPS it was important to look at the X Window System in particular.

The X Window System was developed by the Massachusetts Institute of Technology's (MIT) Athena Project and Digital Equipment Corporation

(DEC) in the mid 1980's. Early versions suffered from compatibility problems with their upgrades, but with the issuance of Version 11, a stable baseline finally emerged.

The X Window System can be considered a standard window system, primarily due to its implementation of a common graphics language. This aids in providing portability for an interface. As a result, all computers have the capability of hosting an interface. In other words, interfaces are not limited to use by any one manufacturer's machines. This directly translates to X Window System's ability to operate well on a network, in a heterogeneous workstation environment.

As mentioned, the X Window System is a "network-transparent client-server based" [1] window system. This allows for users to have easy access to applications all across the network without ever needing to be concerned with how the network implements remote access or execution on heterogeneous machinery.

X Window System also offers an interface designer versatility, for it does not endorse any one particular interface style. This allows a designer the opportunity to create and apply any desired style.

Window systems have one distinctive problem; that being a reliance on low level routines which can be rather tedious to program. Toolkits, on the other hand, provide higher level programming through combinations of low level routines. Thus, toolkits provide routines which are less cumbersome to work with. Some of the various toolkits available include OPEN LOOK Intrinsics Toolkit (OLIT), OSF/Motif Widget Set, and the X Toolkit.

OLIT was developed by AT&T and Sun Microsystems. OLIT is based on Xt Intrinsics and the X Window System, and is written in the C programming language. Thus an application would be dependent on the X Window System, Xt Intrinsics, and OLIT.

The OSF/Motif widget set was developed by the Open Software Foundation. As with OLIT, the OSF/Motif widget set is also based on the Xt Intrinsics and X Window System. The following diagram indicates this.

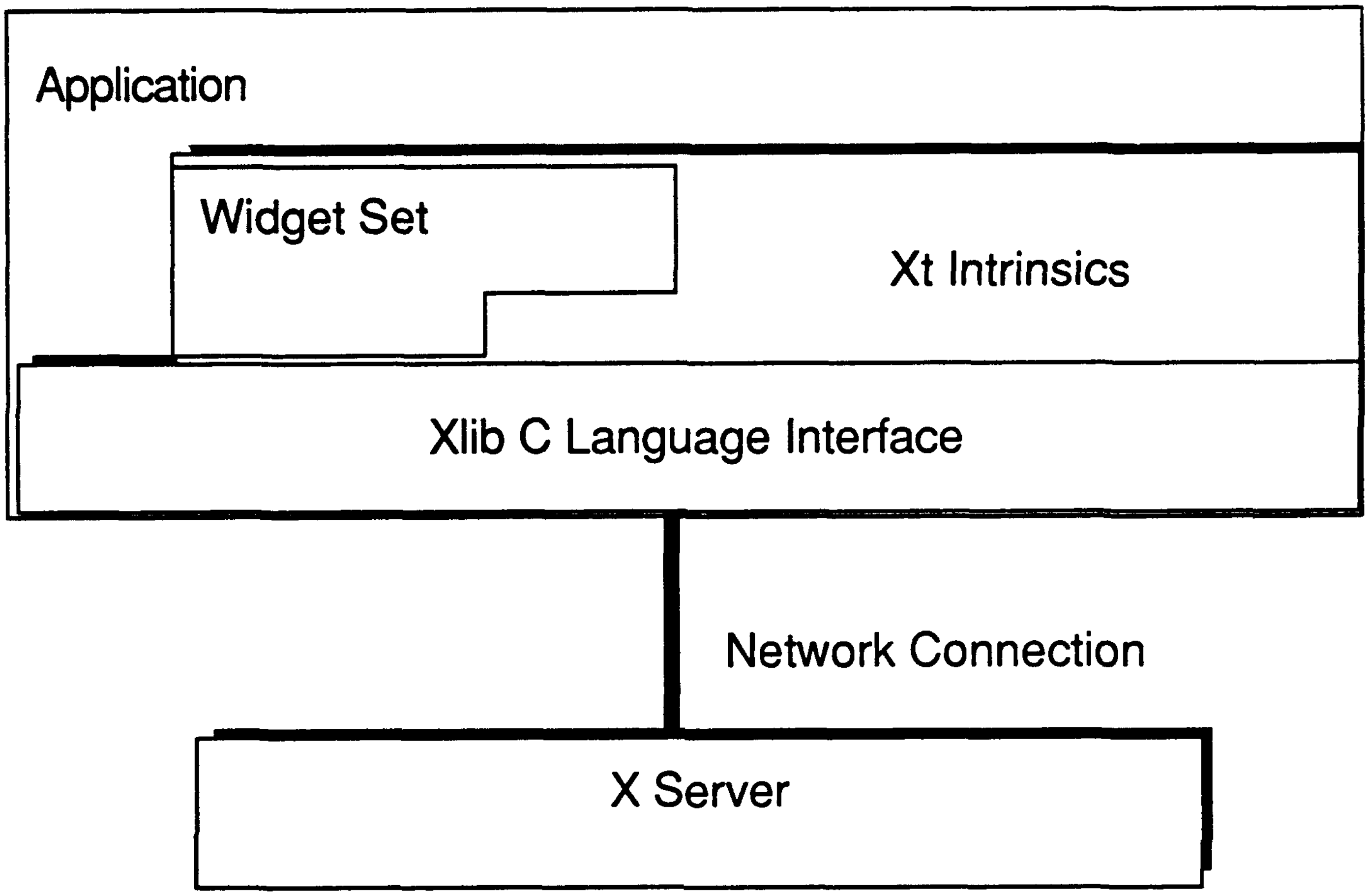

Diagram 1: X Window architecture with OLIT and OS/Motif

OSF/Motif also includes the User Interface Language (UIL) which is "a user interface language which can be used as an alternative to the C programming language" [2].

The X Toolkit includes Xt Intrinsics as well as the Athena Widgets. This toolkit is written in the C programming language, and was the toolkit chosen to implement a prototype of MUSIC for the IPS.

3.0 IMPLEMENTATION OF THE PROTOTYPE MUSIC SYSTEM

3.1 X Toolkit

Because of its maturity, community acceptance and ease-of-use, the X Toolkit was used to implement a prototype MUSIC system. Specifically, the Xt Intrinsics provide high level routines for developing user interfaces. Also, there is a set of user interface components known as the Athena Widgets, which include scroll bars, menus, dialog boxes, and buttons.

The creator of a user interface acquires increased versatility because the toolkit is built from the X libraries. This means that programmers not only can use the Athena Widget set, but can also create new widgets which address the demands of the application.

Also, a programmer does not have to exclusively rely upon the low level routines found in the X libraries of the X Window System. The Xt Intrinsics package furnishes developers with high level routines which are built from routines found in the X libraries, while also allowing for the data type 'Widget'. As a result, there are higher level routines that can be supplemented by low level routines when needed.

3.2 Files Involved

In implementing a prototype for MUSIC, two main files were created; those being the resource file and the actual program file - the application.

3.2.1 Resource File

A resource file contains specific values for various resources. A resource here can be defined as a "named, settable piece of data in some data structure" [5]. In regards to the X Toolkit, these resources generally apply to widget attributes. A few examples of those are the x and y coordinate positions, and the background and foreground colors. There are four types of resource files. They are the applications defaults file, peruser application defaults file, user defaults file, and per-host defaults file. The per-host defaults file should never be modified by the application's default file. These files are parsed by the application in the order they were listed and, as a result, the applications defaults file has the least precedence, while the per-host defaults has the highest precedence. Also, application default files should allocate values as generically as possible in order to increase the ease with which users can customize their own environments.

The following (Example 1) is an example of part of the IPS' application defaults file:

```
Icarus*IcDemo.background:                          PeachPuff4
Icarus*IcDemo.borderColor:                         PeachPuff1
Icarus*IcDemo.label:
PROTOTYPE\n\
\n\
DEVELOPMENT
Icarus*IcDemo.fromVert:                    IcLabel
Icarus*IcDemo.ShapeStyle:
RoundedRectangle
```

Example 1: Resource file

Resources are very useful for two main reasons. First, by establishing resources, a uniform interface can be used to set and inquire about the resources' actual values. Thus all widget classes can be handled the same; for example, in respect to creation and attribute modification. This is accomplished through commands such as XtCreateManagedWidget and XtGetValues, which pass resources to, and also receive resources from, the application. Second applications can thus not only be customized by users to reflect their needs and desires, but also by software installers to meet users' and other specifications.

Resources do not have to be located only in a resource file which is separate from the application, but can also be placed in an argument list, found directly within the application. An argument list can be considered "an array of Arg structures" [5]. There are several advantages to assigning values to resources in this manner. First, resources can be set during the execution of an application (unless a resource file is changed directly before every execution where the values in the resource file are no longer valid). Second, an argument list is useful to application developers specifically, for it provides them with a method to set resources in such a manner that users cannot modify them (the reason will become obvious in a subsequent section). Thus it is possible to provide a certain level of consistency throughout a network by prohibiting users the access of a predetermined set of resources.

The following (Example 2) is an example of an argument list used in prototyping MUSIC:

```
void init_screen(w)
Widget  w;

        {
        Arg          wargs[20];
        int          n;
        int          min_height, min_width, max_height,
                       max_width, width, height;
        int          display_height, display_width;
        int          screen_num;
        Display       *display;

        display = XtDisplay(w);
        screen_num = DefaultScreen(display);
        display_width = DisplayWidth(display, screen_num);
        display_height = DisplayHeight(display, screen_num);
        height = display_height - 100;
        width = display_width - 100;
        min_height = display_height - 350;
        min_width = display_width - 350;
        max_height = display_height - 50;
        max_width = display_width - 50;
        n = 0;

        XtSetArg(wargs[n], XtNwidth, width);      n++;
        XtSetArg(wargs[n], XtNheight, height);            n++;
        XtSetArg(wargs[n], XtNminWidth, min_width);             n++;
        XtSetArg(wargs[n], XtNminHeight, min_height);           n++;
        XtSetArg(wargs[n], XtNmaxWidth, max_width);             n++;
        XtSetArg(wargs[n], XtNmaxHeight, max_height);           n++;

        XtSetValues(w, wargs, n);
        }
```

Example 2: Argument list

These resources provide the width and length dimensions for the introductory screen of MUSIC. Since monitors vary tremendously, this method increases portability. This argument list allows for the width and height dimensions to dynamically change according to the specifications of the monitor currently displaying the interface. Also, users do not have an opportunity to interfere in this aspect of portability for they cannot override those values set in the argument list.

Thirdly, there are situations where creating a resource file can be more of a bother than it's worth. An example would be the temporary use of a minor program that will most likely not be kept for any significant length of time; e.g., a short test program.

Compared to argument lists, resource files are usually the preferred method of declaring resource values for a variety of reasons. First, they provide users with more opportunities to customize an application. Since an application reads the argument list, then the resource file, and lastly the default values for a resource class, and "only takes the first value found" [5] for any given resource -- ignoring all later values found -- the user cannot reset any resources through the resource file that were already set in the argument list. As a result, argument lists limit the resources that a user can alter. Second, software installers have the advantage of not needing to change the actual code -- the argument lists -- for they just have to change the values found in the resource file. Thirdly, applications have the ability to access the resource values from the resource file more efficiently than from the argument list.

An X based program uses the directory /usr/lib/X11/app-defaults as the default path when searching for the resource file. There is no need to include the resource file as a library. While actually developing an application, however, the developer can store the resource file in the directory she/he is working in, and use xrdb (an X server resource database utility) to make sure the contents of the resource file are loaded in the RESOURCE_MANAGER property of the root window. This was done in our implementation, where the working directory was /jet/ing/Icarus.

3.2.2 Application File

The other file created was the application file. In general, an X Toolkit application performs the following few steps: intrinsic initialization, widget creation, callback and event handler registration, widget realization, and event loop processing. Using these steps, a modeless graphical user interface -- the Icarus Presentation System (IPS) -- was created.

As mentioned, the IPS is to have two basic functions. Its primary function is to act as a graphical user interface to access all software systems hosted now and in the future in the ICARUS Facility. The following (Table 1) lists all those currently-resident systems which MUSIC is allowed to access.

Name (full, followed by acronym)	Purpose of System
Non-Cooperative Target Identification (NCTI)	An integrated modeling environment for performing NCTI analysis.
Low Observable Design Synthesis Tool (LODST)	Aero coefficients and mass properties of arbitrary defined geometries are generated.
Electronic Intelligence Tutor (ELINT)	Allows on-line ELINT training and testing.
IVIEW	Graphically displays history files generated by various Air Force simulations.
Tac Brawler (TB)	Air Force standard air engagement simulation.
Electromagnetic Antennae Modeling (EAM)	Integrated software system for performing antenna modeling.
Threat Assessment Support Environment (TASE)	Prototype open systems architecture for intelligence processing and analysis.

Table 1: Current-resident systems hosted by the ICARUS Facility hardware

The other purpose is to act as an interactive briefing tool which will be implemented as the InterVu system. Currently the option is selectable, but is not yet functional.

The application, as of now, is portable to all of the ICARUS Facility hardware and can be remotely executed on any of its hardware. The hardware currently in the ICARUS Facility is shown in Table 2.

Workstations
Silicon Graphics IRIS 4D/85GT
SUN SPARCserver630MP
SUN SPARCstation2

Table 2: Hardware in the ICARUS Facility

The IPS, as implemented to date, consists of a few screens; one of which is the introductory screen. This screen offers three selections, while also informing the user of the interface's purpose. As the user moves the cursor around the screen via a mouse, only the three selections are highlighted; those being Prototype Development, Demonstration Facility, and Quit. In order to proceed with a selection, the user has to release the mouse button within the selection's area. If the user presses the first mouse button down while within the button on the screen, but moves out of the button's area, then the attempt to make a selection is voided. The first selection, Prototype Development, is currently not fully functional. In the future it will serve as a programmer's entrance to the model development activity he or she is interested in working on. The second selection, Demonstration Facility, pops up a screen where each currently available capabilities demonstration can be selected through buttons on the screen. The selecting is done the same way as in the introductory screen.

Two main aspects used to facilitate only a basic implementation of the IPS were a breadth-first approach and stubbing. A breadth-first approach was implemented in order to attempt to demonstrate what a full-up system would look like to the user. This approach calls for the breadth-wise development of each level without going deeper until the entire level is developed. For this prototype implementation, however, no branch is fully developed, and as a result stubbing had be to used. Each option selected that has not been implemented responds with the message:

"This application is not available as of now.
Please exit and make another selection."

Thus this application is a one step interface to all the software systems and demonstrations in the ICARUS Facility. The users are provided with an interface that they are familiar with and that can also easily access systems by the manipulating of screen objects. There is no need to know the actual network's configuration.

4.0 RECOMMENDATIONS

What lies ahead for the IPS? Currently the future plans for the IPS involve the following:

1. employing other input methods such as other Rome Lab prototypes in voice recognition and Virtual Reality devices;
2. implementing the IPS as an iconic user interface;
3. connecting all the software systems in the ICARUS Facility; and
4. building context sensitivity for InterVu.

An example of iconifying the IPS would be the use of a vugraph (or megaphone) symbol as an icon to access the InterVu system.

5.0 SUMMARY

This paper has concentrated on various research and development aspects of the Icarus Presentation System; specifically, on the creation of the GUI aspect of the IPS. It consists of various screens which offer entrance to all the software systems in the ICARUS Facility, MUSIC, and an interactive briefing tool; the InterVu system. Through current and future development, enhancement, and research, the IPS will become a state-of-the-art interface, allowing multi-media access to the systems, as well as audience driven interactive presentations and demonstrations.

BIBLIOGRAPHY

1. Young, Douglas A. and John J. Pew. *The X Window System: programming and applications with Xt.* New Jersey: Prentice Hall, Inc., 1992.

2. Young, Douglas A. *Object-Oriented Programming with C++ and OSF/Motif.* New Jersey: Prentice Hall, Inc., 1992.

3. Yip, Stephen W. L. and David J. Robson, "Graphical User Interfaces Validation: A problem analysis and a strategy to solution", Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences, 1991, p. 91 - 100, vol. 2.

4. Thimbleby, Harold. *User Interface Design.* New York: ACM Press, 1990.

5. Asente, Paul J. and Ralph R. Swick with Joel McCormack. *.X Window System Toolkit: The Complete Programmer's Guide and Specification.* Digital Press, 1990.

6. Nye, Adrian. *Volume One: Xlib Programming Manual.* Sebastapol, California: O'Reilly & Associates, Inc., 1990.

7. Risley, Margot J. and Alex F. Sisti. "Designing a User-Friendly Interface for the Forward Area Processor (FAP) Simulation." RADC-TM-85-14, Jan. 85.