BUILDING INTERACTIVE APPLICATIONS WITH THE X WINDOW SYSTEM

THESIS

Presented to the Graduate Council of
Southwest Texas State University
in Partial Fulfillment of
the Requirements

For the Degree of

Master of SCIENCE

By

Sreedhara Bhasin

San Marcos, Texas
May, 1994

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. John Durrett for his support and guidance during this work. I would also like to thank Dr. Davis and Dr. Ogden for their time and consideration as committee members on this project. I would also like to thank all my teachers in the Computer Science department. I have learnt a great deal from them.

I owe my unending love and gratitude to my mother, who has been an inspiring force in all my learning experiences and who has taught me the value of knowledge.

If it were possible, I would be sharing my degree with my husband, who has worked as hard as I have, to get me through all of this. He has given constant help, endless encouragement and unfailing support and love at every step of the way. I could not have done it without him. I owe him my deepest love and appreciation.

Sreedhara Bhasin

April 18, 1994

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Page

CHAPTER 1

DESIGN ISSUES FOR BUILDING INTERACTIVE APPLICATIONS

1.1 Design Guidelines for Interactive Systems

Computer systems have become pervasive in our society. The increasing use of computers by people from all walks of life, has led to the rising need for easy-to-use, interactive computer applications. Most computers have an interface, through which the user and the computer interact. This interface is often the determining factor in the success and failure of a particular system.

The 90's have been a glowing example of interactive applications design and elegant implementations of such designs. An interactive application essentially promotes in the user a feeling of active participation in controlling the communication between him/her and the computer. This involves on the part of the human, information processing and input operations in response to sensory reception of external stimuli forwarded by the system. The user uses his limbs, his sight and certain mental faculties to interact with a computer interface that not only offers opportunity to communicate with the computer in a highly visual manner, but also offers quick response to user actions, i.e., getting a choice of actions by merely pressing one key.

These days the term "interactive" system has become synonymous with "user-friendly" system. As is obvious from the nomenclature, a "user-friendly" system is essentially interactive. But one has to consistently and methodically design the interface,

in order to incorporate some fundamental axioms of "Human Factors," to make the system truly interactive.

McCormick has defined Human Factors as "Human factors discovers and applies information about human behavior and abilities, limitations and other characteristics to the design of tools, machines, systems, tasks, jobs and environments for productive, safe, comfortable and effective human use."

An interactive system inherently has to take into account the human attributes of the user. In this study, an effort will be made to explore the most significant issues that influence the design of interactive systems and the techniques of interaction that facilitate "friendly communication" between the user and the computer. Painstaking research has been undertaken in recent years to develop a set of ground-rules for building and enhancing friendly computer interaction. It should be noted, that for the purpose of this study, a few techniques and interaction styles have been explored in detail, with example applications/case studies illustrating the effective implementation of them.

Who is a user? A difficult question to answer, considering that computer users include high school children as well as scientists at NASA, thanks to the invasion of the personal computers or the PCs. A general criterion of a friendly application is therefore its versatility to reasonably cater to experts as well as beginners.

Some common-sense guidelines can be evolved.

- A good interactive system is one, that is simple, projecting a "natural" and uncomplicated image of the system.
- Such a system should be responding meaningfully to user actions.
- Such a system should allow actions to be initiated and controlled by the user.

- A friendly system should also have flexibility in directing user actions and display a certain degree of tolerance of errors.

- Of course, a very important aspect of such an application is that it should be stable and be able to detect user difficulties. A good system never 'deadens' the user, that is to leave the user with no recourse at all.

User satisfaction is another important issue. A truly user-friendly system has to give the user the sense that he/she is in control. The computer should appear as a tool under the direction of the user. Many of us still recoil in horror when we remember our very first experience of being a computer user. The point is, a friendly system should relate as closely as possible to individual human capabilities and a varying knowledge base.

One cannot discuss interaction without reference to the concept of compatibility. There are three broad categories of compatibility.

- Conceptual. Conceptual compatibility relates to how meaningful the codes and symbols depicted on the computer screen, are to the people who use them.

- Movement. An extremely important factor is movement compatibility in an interactive environment. It is significant how displays and controls move in response to user actions.

- Spatial. Spatial compatibility refers to the physical arrangement in space of controls and displays.

While developing interactive applications as case studies for this particular study, an attempt has been made to select commonplace themes that can be understood by non-computer professionals and that inherently need to be user-friendly for serving their functionalities.

Conceptual compatibility has been attempted through a 'natural' and distinct system of codes and display. Movement compatibility has been tried by designing related displays and controls such that there is reasonable correspondence to their physical features and asserted goals and perhaps also their modus operandi. Spatial compatibility has been maintained in a broad sense, by visually appealing displays, logically arranging physical groups of choices and information.

1.2 Essential Elements of Friendly Software Design

At the very outset of designing a user-friendly application, it is imperative to examine the essential elements of friendly software design. The golden rules follow:

Know the user. Every software is aimed at a certain group of users. It is invaluable to identify that group and their degree of computer literacy.

Communicate visually. The explosion of GUI or graphical user interface lately, speaks volumes about the glory of this particular aspect. Some of the glowing examples are the Microsoft Window and the X Window System. Graphical images have become the most powerful tool for conveying a "action-response" based interaction that has offered a tremendous impetus to user-friendly software usage and development.

Psychologists have long discovered that spatial relationships are grasped more quickly than linguistic representation. Users experience less anxiety because the system is comprehensive and it is easy to change the direction of activity. There is little need for mental decomposition of tasks into multiple commands with complex syntactic form.

Speak the user's language. For an application to be comprehensible, the language of interaction has to be entirely understood by the user. Jargon from the user's dialect can communicate effectively. Jargon from the programmer's dialect should be avoided.

Reduce the user's defensiveness. Computer programs argue with their users all the time. They shout back ominous warnings like "Illegal Command," "Irrevocable application error," with the ferocity and the stubbornness of a child. This makes the user feel helpless, angry and confused.

As a palliative for this situation, it is the duty of the designer to make the communication descriptive, instead of judgmental, make the messages explanatory, rather than cryptic.

Respond to the user's action. By making the effect of the user's action immediately visible, programmers can influence the users to grasp a "walk-through" through the application rather easily. Quick response tends to make a program seem simple and obvious.

Help the user crystallize his thoughts. Many users have a fuzzy idea about what a piece of software really does. Friendly software should present itself to users in a way that helps them recognize what they want to do, instead of requiring them to spend time formulating choices for their course of action. Graphical interfaces alleviate much of these difficulties, by presenting the user with well-defined routes through a window and/or menu based hierarchy of commands.

Make the design simple. Simplicity is the magic word. A simple system is a friendly system, since it does not pose the inhibiting threat of traversing a labyrinth of complicated commands. A simple design introduces clarity and effectiveness, hence making it easy for even a beginner to navigate his way through.

Throughout this endeavor, the underlying paradigm for writing interactive programs has been "simple, consistent and efficient." To make any program user-friendly, we need to retain consistent sequences of actions in similar situations and also offer feedback.

1.3 Techniques of Interaction

In this context, it is useful to mention another aspect of writing interactive applications. There is a whole range of interaction styles or techniques that can be readily used. This study would adopt and implement most of the general categories of interaction styles. They are briefly mentioned here. Their use and implementation would be evident through the process of building the interactive example programs.

Natural language interface. Gives the user the ease of communicating with the computer in a natural language, which is really a well-defined subset of some natural language like English.

Menu based interface. Pages can be written extolling the virtues of a menu based interface. Menus present the hierarchy of choices in a visual and distinct fashion , thus making it very simple for the user to take the desired course of action. It eliminates training and memorization of complex commands.

Form filling. The form filling dialogue is one, in which the user issues commands by filling in fields in one or more forms displayed on the screen. This is a very useful style for allowing the user to input large amount of data, since the form design most often contains, demarcated entry and display fields and also possibly, available choices.

Iconic interface. This is the type in which the user commands and system feedback are expressed in graphical symbols and/or pictograms.

Direct manipulation. One of the most vital elements of interactive interface is the capability to directly manipulate the object of interest through a variety of pointing devices. Direct manipulation has been greeted by the users with glowing enthusiasm. The central idea is the visibility of the objects and rapid, reversible and incremental actions.

Graphical interaction. In a graphical interface the users are primarily manipulating two-dimensional and three-dimensional images. Graphical interface has been the biggest

success story in recent years. The windowing system has become central to graphical interfacing techniques. GUI (Graphical User Interface) incorporates a variety of graphical representations of commands, structures and other data. Windows, menus, icons are all very well suited for a graphical interaction style.

This study is committed to the idea that the X Window System, which is itself a graphical system, is very effective for building user-friendly applications. In the following chapters the X Window System would be explained in detail and its suitability and appropriateness for being used as the medium for building interactive software would be amply explored.

There are many more aspects of developing 'amicable' systems. There are design issues to consider and balance properly. Attention has to be paid to matters of data display, data entry system, dialog design, messages for prompting and error correction. A good interactive system is a right combination of several elements. The primary idea is that a computer application should serve the human in a convenient and comfortable manner.

This thesis attempts to draw attention to the most basic requirements for designing user-friendly software and tries to bolster the premise that the X Window System is extremely effective for building such applications. As part of this exercise, an attempt will be made to familiarize the reader with the rudiments of the X Window System. Furthermore, applications developed as case studies would be used for demonstrating the building process in an X environment.

CHAPTER 2

X WINDOW SYSTEM CONCEPTS

The sole purpose of this chapter is to offer an overview of the X Window System. The following discussion explores the understanding of the X Window System or X Windows, as it is commonly called, and explains what it means in practical terms, for a user, a programmer and an application developer. The concepts that went into the design of the X Window System are briefly outlined. We then look at a couple of basic example programs written using X.

To the user, the X Window System is a set of windows. A window consists of a configurable frame with a set of configurable options. Each window is an independent entity and several windows can coexist at the same time. If a window under another window is brought to the top, its contents appear unsmeared.

From a programmer's point of view, the X Window System is slightly more than it is to the user. The programmer understands that the system consists of a server that has the capability to open windows, maintain their hierarchy, allocate desired colors and allow graphics and text to be drawn into the windows. The program the programmer must write is part of a client that communicates with the server.

The server returns input from a user in the form of events, which he/she must use to take appropriate actions. For example, if a covered window is exposed, its contents must be redrawn, so that it appears as if the original contents were never lost. Similarly,

mouse and keyboard interaction must be handled. Finally, the programmer understands that there exists a window manager that controls the behavior of the windows, and how to interact with them.

To an applications developer, the X Window System consists of an extremely portable platform on which to develop highly user-friendly applications. The windowing system provides the basic framework for point and click user interaction, a direct manipulation technique. The graphics capabilities provide the facilities needed for the components, such as menus, panels, dialog boxes etc. In addition, the existence of advanced toolkits that contain predesigned widgets makes designing user-friendly applications a lot easier and faster.

Let us look at the structure of the X Windows System in detail. It is important that we go into some level of technical detail to understand some of the concepts.

2.1 The X Window System

The X Window System is a network based, asynchronous client-server system, that communicates via information packets over a network. It provides a hierarchy of resizable windows and supports high performance, device independent graphics. Unlike, most other window systems for UNIX that have a built-in user interface, X is a substrate on which almost any style of interface can be built. On one side of the communication link is the client, which makes windowing, graphics and input commands. On the other side is the server which keeps a complete data structure list and implements the commands sent by the client.

2.1.1 The Server and the Client

The server is the software that manages one display, keyboard and mouse. One user is controlling the keyboard and mouse and looking at the display controlled by the server. The client is a program displaying on the screen and taking input from the keyboard and the mouse. A client sends drawing requests and information requests to the server. The server sends back to the client user input, replies to information requests and error reports. The client may be running on the same machine as the server or on a different machine over the network.

The X window System is not limited to a single client interacting with a single server. There may be several clients interacting with a single server, which is the case when several applications are displaying on a single screen. Also, a single client can communicate with several servers, which would happen when an announcement program is displaying the same thing on several people's screen.

The window manager is a client that has authority over the layout of the windows on the screen. Certain X protocol features are used to enforce this authority. Otherwise, the window manager is just like any other client.

X clients are programmed using various client programming libraries. The C libraries are the most widely used. They include a low level procedural interface to the X protocol, called Xlib and a higher level toolkit written in an object oriented style called the Xt Intrinsics. The Intrinsics are used to build user interface components called widgets.

Xlib and Xt Intrinsics were developed at MIT. Several other toolkit layers that use Xlib interface to the protocol have been developed since. This study would use Motif.

2.1.2 The X Protocol

The X protocol is the true definition of the X Window System and any code in any language that implements it is a true implementation of X. It is designed to communicate all the information necessary to operate a window system over a single bi-directional stream.

2.1.3 Message Types

The X protocol specifies four types of messages that can be transferred over the network. Requests are sent from the client to the server and replies, events and errors are sent from the server to the client.

- A request is generated by the client and sent to the server. A protocol request can carry a wide range of information, such as specification for drawing a line or changing the color value in a cell in a color map or an inquiry about the current size of a window.

- A reply is sent from the server to the client in response to certain requests. Not all requests are answered by replies - only the ones that ask for information. Requests that specify drawing, for example, do not generate replies, but requests that inquire about the current size of a window do.

- An event is sent from the server to the client and contains information about a device action or about a side effect of a previous request.

An error is like an event, but it is handled differently by clients. Errors are sent to an error handling routine by the client-side programming library.

A protocol request that requires a reply is called a round-trip request. These have to be minimized in client programs because they lower performance.

2.1.4 Division of Responsibility

The server is designed to, as much as possible, hide the differences in the underlying hardware from client applications. The server manages windows, does all drawing and interfaces with the device drivers to get keyboard and pointer input. The server also manages off-screen memory, windows and fonts, cursors and colormaps.

Having the server responsible for managing the hierarchy and overlapping of windows has few, if any, disadvantages. It would seem possible for the client to waste network time by requesting graphics to a window that is not visible, since the client knows nothing about the window position. However, this situation is dealt with by having clients draw only in response to an Expose event that announces when an area of a window becomes exposed. It is the client's responsibility to send the appropriate requests needed to redraw the contents of the rectangle of a window specified in the Expose event.

Some decisions were made to purely simplify client programming. For example, coordinates in drawing requests are interpreted relative to the window being drawn into, rather than to the entire screen. This provides a virtual drawing surface and makes client programming easier, because the client does not have to continually track the window position and calculate where to draw. This gives the server the burden of determining where to actually draw graphics on the screen based on window positions.

In other cases, decisions were made to increase performance. An example of this is the Graphics Context, a feature of X that is invaluable to the business of building highly graphical user interfaces and hence, interactive applications. The X Graphics

Context (GC) allows the server to cache information about how graphics requests are to be interpreted, enabling a compact and very clean way to modify and reconstruct visual elements of an application. Also, this caching of GC makes possible easy and quick switching between them. It is also a happy coincidence that GC usually makes client programming easier, because it reduces the number of parameters needed in drawing calls.

The GC is one of the several abstractions X maintains in the server, the most important others are the window, colormap, pixmap and font. The client refers to each abstraction in protocol requests using a unique integer id assigned by the server.

2.2 A Sample Session

The following describes what happens over the network during a minimal application that creates a window, allocates a color, waits for events, draws into the window and quits.

The network events that will take place during a successful client session are:

- Client opens connection to the server and sends information describing itself.
- Server sends back to client data describing the server or refusing the connection request.
- Client makes a request to create a window. This request has no reply.
- Client makes a request to allocate a color.
- Server sends back a reply describing the allocated color.

- Client makes a request to create a Graphics Context for using in later drawing requests.
- Client makes a request identifying the types of events it requires. In this case, Expose and ButtonPress events.
- Client makes a request to map the created window.
- Client waits for an Expose event before continuing. This sends the accumulated requests to the server.
- Server sends to client an Expose event indicating that the window has been displayed.
- Client makes a request to draw a graphic, using Graphics Context.
- Loop back to wait for an Expose event.

Client requests are queued up by the client library before being sent to the server and the client reading events triggers the sending. This is not a required characteristic of client libraries, but it improves performance greatly because it takes advantage of the asynchronous design of the protocol. Xlib works this way. It allows the client to continue running without having to stop to wait for network access, until it would have to wait for an event anyway.

Many of the actions taken by the client in this session must be done in the order shown for the application to work properly. For example, Expose events must be selected before the window is mapped, because otherwise, no event would arrive to notify the client when to draw. This becomes even more important when a window manager is managing the screen. Many window managers let the user decide on the size and position of the window before allowing it to be mapped, introducing a sizable delay between the

client requests to map the window and when the window actually appears on the screen ready to be drawn into. Only the Expose event tells the client when it is time to draw.

Colors are allocated very early in the session before creating the graphics context to optimize the usage of protocol requests. To allocate a color, the client tells the server what color is desired and the server responds by giving the client a pixel value which is a number that identifies a cell in the colormap containing the closest color available. When creating the Graphics Context, this pixel value can be used to set the foreground color to be used for drawing.

The client allows the user to identify the server it wants to connect to by specifying a host and a display number. The display number is zero for personal workstations because there is only one keyboard, pointer and display connected to a single host, and hence, only one server.

The client side library should provide an easy-to-use method for the user to specify which server to connect to. Under UNIX, Xlib reads the environment variable DISPLAY. The user specifies the server by setting DISPLAY to the hostname and server separated by a colon, for example Sleepy:0. Once the proper address is known, the client begins sending packets that describe itself.

2.3 Example Programs

It is useful at this stage to look at a couple of example programs which offer an idea about the basic working of the X Window System.

An important thing to remember is that X programs are event-driven, which means that after setting up all the server resources, the program performs all further

actions in response to events. The event-gathering loop is the standard way to respond to events, performing the appropriate action depending on the type of event and the information contained in the event structure.

The following examples can be used as a very simple and basic model of a X program. Complicated and sophisticated applications can be built by extending this very basic framework of event-driven programming. As is the experience of this author, a beginner can start with these very programs and then add more code for additional functionality of their programs.

The protocol, as we will see, is completely hidden from anyone using or programming with X. Applications communicate with the server by means of calls to a low-level library of C language routines known as Xlib. Xlib provides functions for connecting to a particular display server, creating windows, drawing graphics, responding to events and so on.

The window manager is a special type of client written with the X library. By convention, it is given special authority to control the layout of windows on the screen. Clients usually do not control the location of their windows on the screen. Instead, they give hints about the size and location of their windows to the window manager. It controls the interaction with users and allows users to move or resize windows, start new applications and control the stacking of windows on the screen. All applications must cooperate with the window manager.

The following programs are written solely with Xlib. The language used is C and the C style of commenting has been used throughout this study. Applications can also be written with toolkits which are a set of higher level subroutine libraries. Toolkits implement a set of user interface features such as menus or command buttons readily.

2.3.1 Function Calls

The major function calls that these programs utilize are explained below.

XOpenDisplay(). This command establishes a connection of the client with the server. The client indicates which server it requests a connection with and the appropriate server, on receipt of this communication, decides if it wants it. If so, it returns to the client a pointer to a structure which contains information about the server and which the client uses in all its subsequent communications with the server.

XCreateWindow(). This command requests the server to create a window. The parameters to this command are:

- The handle to the sever.
- The parent of the window to be created.
- The size of the window to be created; origin x and y, length and height, width of the border of the window.
- The depth of the window, which is less than or equal to the parent's depth.
- Window class. Whether input/output or output only.
- The type of visual.
- Pointer to a structure containing numerous attributes that the window should have.

XMapWindow(). This command requests that the created window be displayed. XCreateWindow creates structures for the window but does not display it. The window is supposed to be in existence now and a lot of manipulation with respect to its attributes, color, cursors, etc. can be done. But the window is displayed on the screen only after this call has been made.

XNextEvent(). This call waits for an event on the particular server. An event is generated due to several reasons, such as, buttonpress, keypress, resizing of a window, exposing a hidden window, changing its colormap, changing the position of a window etc. A detailed list of all the events can be found in any good X manual. A window may not be interested in all the events possible and at the time of creation (or later), it can be specified what kind of events may be reported for the window.

XDestroyWindow(). This unmaps (removes from display) the window and destroys all the data structures associated with the window. Any subsequent references to this window are considered illegal.

XCloseDisplay(). This severs the connection of the client with the server.

2.3.2 Data Types

The X data types that appear here are :

Window: A unique identifier that is returned by XCreateWindow() and is thereafter used by the program to refer to that particular window.

Display: A large structure that contains information about the server. It is filled only after the connection is set up by the XOpenDisplay call.

XEvent: XEvent is a union that has as its first member the type of Event, such as ButtonPress or Expose, and a simple event structure. When an event is received (with the XNextEvent call, for example), the application checks the type and the specific event structure (such as xexpose for Expose event) is used to access information specific to that event.

XSetWindowAttributes: A structure that contains the attributes of a window. To set the window attributes, the elements of the structure have to be set to desired values and a valuemask argument has to be set that represents which members are to be changed in the server's internal structure. These information can be passed while creating a window.

GC: The GC is a structure containing all information about the graphics functions to be used, the colors to be used for drawing, the fonts to be used, line widths, line styles, clipping details and so on.

XGCValues: A structure that keeps information about the components of a new GC.

XColor: A structure that can store the pixel value of a read-only colormap cell

with the closest values.

Colormap: A colormap used as a lookup table stored in the server. Any given pixel value is used as an index into this table.

XPoint: A structure with two short integers as it members.

2.3.3 The First Example

The first program is a simple program that demonstrates the basic actions that any application has to take to create and display a window. Nothing is drawn into it. The concept of events is introduced and the application waits for an event of the "buttonpress" type, on receipt of which it terminates. The program uses the C language and its syntax.

The program follows.

```
/*****************************************************************/
                    /* This program creates an X Window */
```

```
/*****************************************************************/
/* Standard C include files */
#include <stdio.h>
#include <string.h>
/* Xlib include files */
#include <X11/Xllib.h>
#include <X11/Xutil.h>
main()
{
        XSetWindowAttributes values;
        Display *display;
        Window win, root;
        XEvent event;
        int screen;
        int value_mask = 0;
        unsigned long BackColor;
        /* Connects to server and returns a pointer to a structure of type Display */
        display = XOpenDisplay(NULL);
        /* The screen is set to the return value of the DefaultScreen macro and used
        throughout to indicate which screen to operate on */
        screen = DefaultScreen(display);
        /* BlackPixel and WhitePixel are macros that return pixel values
        corresponding to the named colors. Note that applications choose colors and are
        returned pixel values by routines that allocate colors or they get
        pixel values from macros */
        BackColor = WhitePixel(display, DefaultScreen(display));
```

```
/* Each screen has its own root window. To create the first window of an
application, the root window is used as the parent on that screen */
root = RootWindow(display, screen);
/* This program needs to receive the event types of exposure and buttonpress for
its purpose . This program has to select these types of events
specifically */
/* Tell the X server we are interested in ButtonPress and Exposure Events
Each symbol in the event mask selects one or more events. The event mask
constants are combined with a bitwise OR */
/* Set the values for the window attributes */
values.event_mask = ButtonPressMask | ExposureMask;
values.background_pixel = BackColor;
value_mask = CWEventMask | CWBackPixel;
/* Create the window - returns the window id */
win = XCreateWindow(display, root, 0, 0, 400, 400, 0, CopyFromParent,
        InputOutput, CopyFromParent, value_mask, &values);
/* Now map the window */
XMapWindow( display, win);
/* Wait for the Exposure of the window in a while loop. XNextEvent will get an
event structure. Here the Expose event will be received */
while(1)
{
        XNextEvent(display, &event);
        if(event.type = = Expose)
                break;
}
```

```
        /* Block at this point till a buttonpress occurs */
        while(1)
        {
                XNextEvent(display, &event);
                if(event.type = = ButtonPress)
                        break;
        }
        /* Kill the window */
        XDestroyWindow(display, win);
        /* Close connection with the server */
        XCloseDisplay(display);
        exit(0);
}
```

2.3.4 The Second Example

The second program is more detailed. Here is a list of the subroutines it uses.

Xsetup(). This subroutine contains function calls to do the following setup.

- Establish a connection to the server.
- Get the identity of the root window and the screen.
- Set up the events we are interested in for this window.
- Set up the background color of the window.
- Set up the border of the window.
- Set up the color depth.

- Call XCreateWindow with the necessary parameters.
- Call XAllocColor(). It allocates a pixel value of a colorcell. The RGB values of the XColor structure can be set to get the closest color desired.
- Call XCreateGC(). The XCreateGC call sets up a GC for a particular window.
- Ask that the window be displayed.
- Wait till the server replies (by an event) that the window has been displayed.

Xdraw(). This routine contains all the subroutines that draw graphics within the window. These are:

- XDrawString(). This command draws a text string at the location specified. The font and the color used are as per the Graphics Context used.
- XDrawLine(). This command specifies that a line be drawn between the end points specified. The color of the line, its width and its style are as per the GC used.
- XDrawRectangle(). This command draws a rectangle.
- XDrawArc(). This command draws an arc. Arcs are drawn specifying the start and end angles.
- XFillPolygon(). Draws a filled polygon.

Wait_for_mouse_click(). This routine simply waits till the user generates an event of the buttonpress type by clicking any button of the mouse.

Wind_up(). This routine does the cleaning up. All allocated resources by the server must be deallocated before severing the connection, in the interest of conservation and

efficiency. The GC is freed, the window is cleared and destroyed and the connection with the server is severed.

The program follows.

```
/*****************************************************************/
                /* This program creates an X window. It then draws various
                        graphical entities within the window */
*****************************************************************/
#include <stdio.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#define STDEVENTS ExposureMask | ButtonPressMask
/* Points are of the type XPoint */
XPoint Points[] = {{50, 250}, {125, 250}, {155, 300}, {87, 350}, {20, 300}};
unsigned long ForeColor, BackColor;
int screen;
int value_mask = 0;
GC xgc;
XEvent xev;
Display *display;
Window win, root;
XSetWindowAttributes xswa;
XGCValues gcvals; /* A structure which will provide components for the new GC */
XColor xc;
Colormap dcmap;
main()
```

```
{
        /* Create and draw the window */
        Xsetup();
        /* Draw graphics within the window */
        Xdraw();
        /* Wait for mouse click */
        Wait_for_mouse_click();
        /* Close window and sever connection with the server */
        Wind_up();
        exit(0);
}
Xsetup()
{
        /* First set the connection */
        display = XOpenDisplay(NULL);
        screen = DefaultScreen(display);
        ForeColor = BlackPixel(display, DefaultScreen(display));
        BackColor = WhitePixel(display, DefaultScreen(display));
        root = RootWindow(display, screen);
        dcmap = DefaultColormap(display, screen);
        /* Tell the X server we are interested in the events defined as STDEVENTS */
        /* Set the window attributes */
        xswa.event_mask = STDEVENTS;
        xswa.background_pixel = BackColor; /* Window background to be white */
        xswa.border_pixel = ForeColor; /* Border is black */
        value_mask = CWEventMask | CWBackPixel | CWBorderPixel;
```

```
        xswa.border_pixel = 1; /* Border pixel value */
        win = XCreateWindow(display, root, 0, 0, 400, 400, 0, CopyFromParent,
         InputOutput, CopyFromParent, value_mask, &xswa);
        /* Get the desired color . Blue in this case. Scale it by 255 */
        xc.red = 0 * 255; xc.green = 0 * 255; xc.blue = 255 * 255;
        XAllocColor(display, dcmap, &xc);
        gcvals.foreground = xc.pixel; /* Assign the allocated color */
        /* Create The GC */
        xgc = XCreateGC(display, win, GCForeground, &gcvals);
        /* Map the window */
        XMapWindow(display, win);
        /* Wait till the window is drawn */
        while(1)
        {
                XNextEvent(display, &xev)
                if(xev.type = = Expose)
                        break;
        }
}
XDraw()
{
        /* Draw some text */
        /* Takes as parameters the GC, x and y coordinates of the baseline of
        the string and the length of the string */
        XDrawString(display, win, xgc, 15, 50 , "test_string",
        strlen("test_string"));
```

```
        XDrawString(display, win, xgc, 15, 60 , "Some Lines",
        strlen("Some Lines"));
        /* Draw some lines */
        XDrawLine(display, win, xgc, 100, 90, 300, 90); /* Specify the coordinates
        of the endpoints of the line */
        XDrawLine(display, win, xgc, 100, 100, 300, 100);
        /* Draw a rectangle */
        XDrawRectangle(display, win, xgc, 150, 120, 100,75 ); /* Specify the x and
        y coordinates of the upper-left corner and the width and height */
        /* Draw an arc */
        /* Specify width and height of the major and minor axes of the arc.
        Angles are specified in 64ths of a degree */
        XDrawArc(display, win, xgc, 200, 225, 200, 150, 45 * 64, 270 * 64 );
        /* Draw a polygon */
        /* Points specifies a pointer to an array of point type;
        specify the number of points, shape and mode */
        XFillPolygon(display, win , xgc, Points, 5, Convex, CoordModeOrigin);
        XFlush(display); /* Flush the request buffer */
}
Wait_for_mouse_click()
{
        int done;
        done = 0;
        while(!done)
        {
                XNextEvent(display, &xev);
```

```
            if(xev.type = = ButtonPress)
                done = 1;
        }
}
Wind_up()
{
        /* Stop the X process and destroy the graphics context */
        XFreeGC(display, xgc);
        XDestroyWindow(display, win);
}
```

Now that we have shown some simple programs, we can proceed to demonstrate the building process of the first application. The design issues of human factors are intertwined with the mechanics of building an interactive application. The principles of user-friendly interface have to accommodated in a programming environment, in this case, within the scope of the X Window System. The design process and the selection of appropriate programming building blocks as well as the assembling of all the pieces will be demonstrated through several phases. The next chapter follows in that direction.

CHAPTER 3

A SIMPLE APPLICATION

In this chapter, an effort has been made to delineate the building process of an interactive application, but a relatively simple one, for demonstrating the techniques of computer-human interaction. As we have noted earlier, there is a multitude of interaction techniques. Each has a specific purpose, such as to designate a position or select a displayed object, and each is implemented with some device.

An effective interaction design is one in which a user carries out his/her work with minimal conscious attention to his/her tools and maximal effectiveness. Interacting with a computer like all human behavior, involves three types of basic human processes. The first is the power of perception, whereby computer generated stimuli, audio as well as visual, are transmitted to the brain and understood and recognized. Second is the power of cognition, which deals with how human beings acquire, organize and retrieve information. The third is motor activity, which opens up the issue of using one's limbs and sight in a comfortable manner while interacting with a computer.

Keeping in mind the above determinants, it seems obvious that a friendly application has to appear "natural" and intuitive. Naturalness stems from handling input devices that control displays in ways that are analogous to action-reaction in the real world.

3.1 Interaction Tasks

It must be noted that the performance of a computer action generally involves a series of tasks to be carried out almost as a single unit. This is very true when it comes to following computer commands to attain a desired result, for example, traversing a hierarchical menu structure to find a particular choice. While building this application, an effort has been made to naturally group sequential actions into action sequences.

Any application of any order, can be decomposed into some basic interaction tasks. These are essentially independent of application and hardware and form the building blocks from which more complex interaction tasks, and in turn, comprehensive interaction dialogue are assembled. These tasks are user oriented, in that they are the primitive action units performed by a user. It is therefore imperative that a user-friendly application allows scope for graceful integration of these fundamental interaction tasks.

The four types of interaction tasks that have been dealt with in this study are:

- Select
- Position
- Orient
- Text

The applications discussed in this chapter, illustrate effective incorporation of these rudimentary action-tasks into a "natural" looking, simple, computer interface.

Select. The user makes a selection from a set of alternatives. The set might be a group of commands. One such typical interaction technique would be to choose a menu with a pointing device.

Position. In carrying out a positioning task, the user indicates a position on the interactive display, often as part of a command to place an entity at a particular position.

Orient. In this case, the user orients an entity in a two-dimensional or three-dimensional space. This might involve rotating a symbol or an object in a particular manner.

Text. The user inputs a text string or receives a text string in response to an action. Often, the text string itself becomes part of the information stored in the computer, rather than serving as just a command.

3.2 Controlling Tasks

None of the aforementioned interaction tasks, directly modify an object being displayed. A task, which has as its basic purpose, the control of objects already visible on the display, is a controlling task. These are the elementary controlling tasks. The three major types of controlling tasks that have been explored here are:

- Stretch
- Manipulate
- Shape

Stretch. The user grasps a particular feature of an object and moves it to a new position, leaving the remaining features of the object in place. The result is a distortion of the object, much like stretching a rubberband.

Manipulate. The user causes an object to move in the display space, by either translation or orientation under the control of a locator/input device. Scaling, twisting are such manipulation techniques.

Shape. The user can cause an object to change its shape under the control of a pointing device.

It is time to describe the first example application. While designing this application, it has been kept in mind to adhere to the general guidelines of Human Factors essentials. Also, it has been designed such that the fundamental action tasks, both interaction and control, can be included and demonstrated to the user.

The application makes use of a graphical interface, the foremost prompting factor being the very graphical nature of the X Window System. A very significant aspect to note here is the special "free" characteristic of the X Window System. X was purposely designed to be "policy free," meaning that it does not come with any standard interface, like many other windowing systems do. The application designer can write all parts of the user interface himself/herself if desired. He/She can design and write menus, buttons, dialogue boxes and all other interface features and also decide how they are to be used. Thus, a designer can wield a lot more power and enjoy a much greater degree of freedom while designing and programming a X application. The application discussed in this chapter has been programmed using Xlib alone and in the C programming language.

3.3 The Application

The application starts out by displaying a main window. The main window is the primary work area and contains a menubar across the top of the window. The menubar contains labels which indicate the title of the main menus available. Each label is drawn in a different color. In widget terms, each of these labels is a CascadeButton, which is similar to the PushButton except that CascadeButton invokes a pulldown menu or a dialog box and not application code. Each of these menu labels or CascadeButtons brings up a pulldown menu with a certain number of menu items or PushButtons, that invoke certain application functions.

The work area or the frame of the main window displays a polygon shaped like a star, drawn in solid black lines. This is the default state of the polygon.

The four primary menu labels or CascadeButtons are:

1. Type

2. Color

3. Edit

4. Quit

See Figure 3.1.

1. Type: Clicking on the Type menu brings up a pulldown menu with a number of menu items or entries. The pulldown contains a titlebar across the top, drawn in a contrasting color and displays the name of the main menu label that was chosen. The displayed menu items are:

1.a. Poly

1.b. Line

1.c. Point

See Figure 3.2.

Each of the above items are drawn in the color, same as that of the CascadeButton. There is a separator (drawn in a lighter shade) between each item. As the mouse is dragged from one item to another, the choice is highlighted (drawn in a darker shade). The titlebar and the separators are used for all of the menu options.

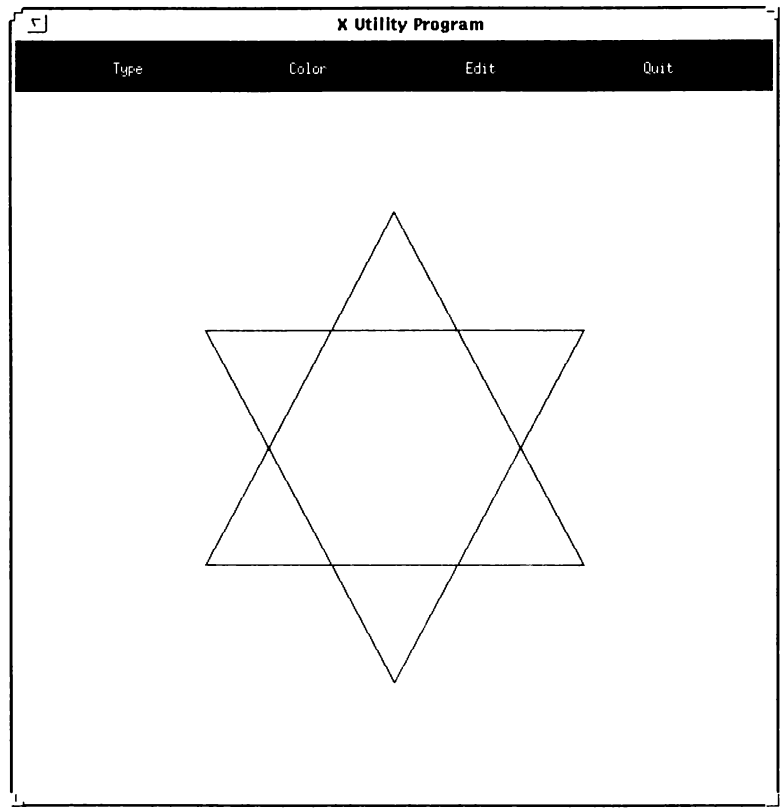

Figure 3.1 The main window

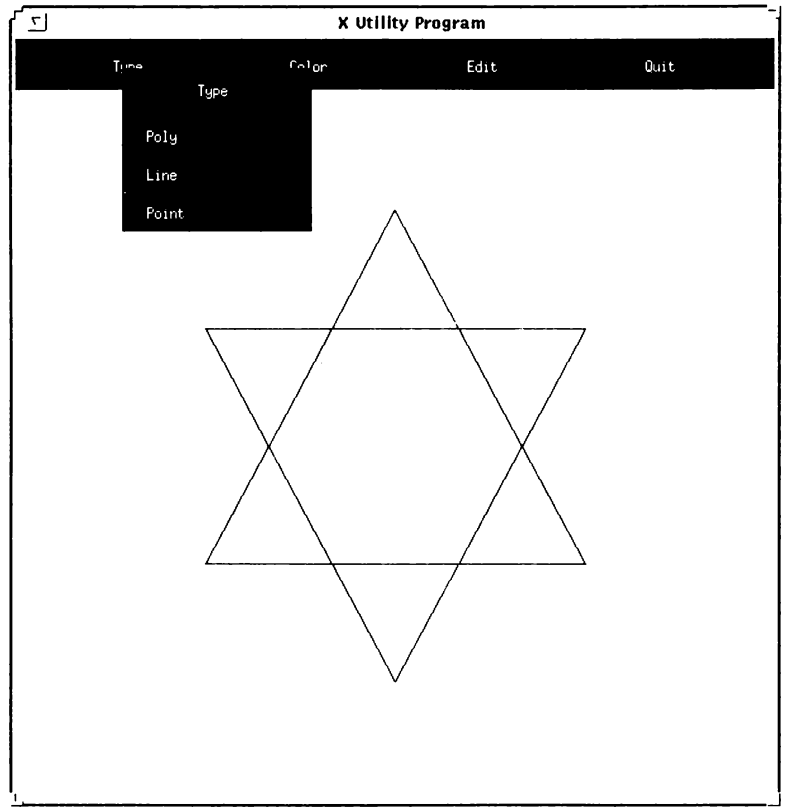

Figure 3.2 The Pulldown menu items for Type option

1.a. Poly: Clicking on the Poly menu button transforms the polygon (the star) into a filled Polygon.

1.b. Line: Clicking on the Line button draws the polygon in solid lines, which is also the default.

1.c. Point: Clicking on the Point button redraws the polygon such that, only the vertices are the visible points.

2. Color: Clicking on the Color button brings up the following:

2.a. Color Palette: A vertically aligned row of color rectangles or a colorbar. Each of the bars represents that very color as a choice. As the mouse is dragged from one colorbar to another, the selected color is highlighted and the name of that color is displayed. There are eight color choices. Selecting any of the color, redraws the polygon (in its current shape) in the chosen color.

3. Edit: Clicking on the Edit button brings up a pulldown menu with the following menu items.

3.a. Scale

3.b. Rotate

3.c. Stretch

3.d. Reset

See Figure 3.3.

3.a. Scale: If the Scale menu is chosen, the scaling action begins. The polygon, in its current shape and current color, is scaled up in size and redrawn in constant motion. At the same time, a narrow prompt or message window appears across the bottom of the

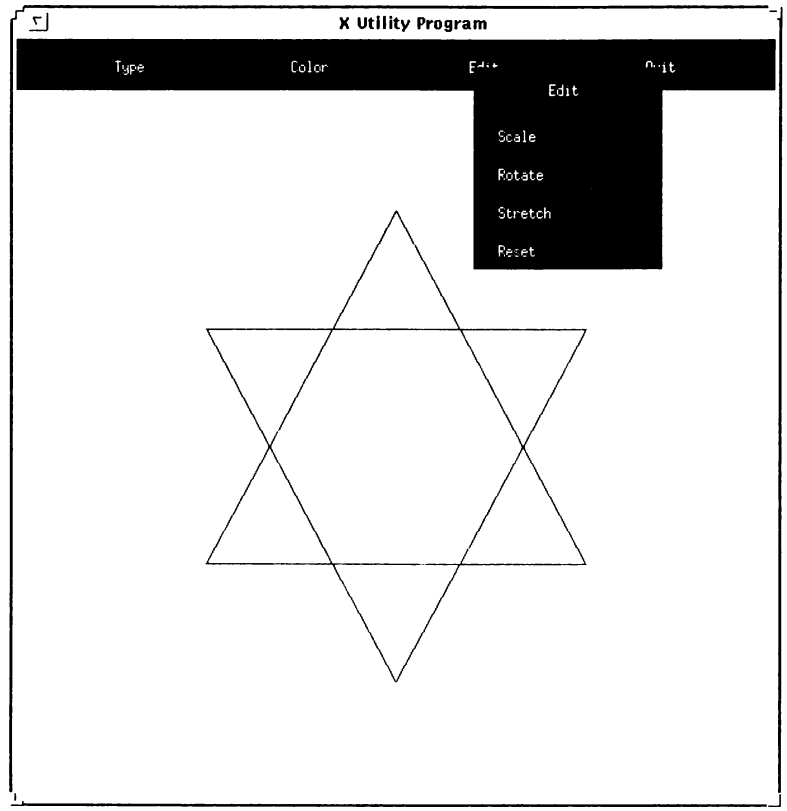

Figure 3.3 The Pulldown menu items for Edit option

main window with the following message; "Scaling: Press middle button to stop." If the middle mouse button is pressed, the scaling action is terminated and the polygon is frozen in its current state. Selecting the Scale menu again, starts another scaling action, this time scaling the polygon down in size. The scaling action can be stopped in the same manner as before.

See Figure 3.4.

3.b. Rotate: Selecting the Rotate menu begins a rotating action. The polygon, in its current shape and color, begins to rotate in a clockwise motion. At the same time, a narrow prompt or message window appears across the bottom of the main window with the following message; "Rotating: Press middle button to stop." If the middle mouse button is pressed, the rotating action is terminated and the polygon is frozen in its current state.

3.c. Stretch: Selecting the Stretch menu brings up the prompt window with the message; "To stretch: Select a vertex by clicking left button of mouse on it. Drag mouse to stretch it." If the button is clicked on a vertex of the polygon (approximately within a range of ten pixels), another message appears in the prompt window; "Vertex found. Drag mouse with left button pressed to stretch. Release button to stop." If done so, the rubberbanding action begins. The edges connecting the chosen vertex gets redrawn in dashed lines and continuously follow the point as it gets dragged by the mouse to any other location. As the mouse button is released, the vertex point is frozen and the edges are redrawn joining the vertex at the new location.

If the user is unable to select a vertex (clicks at a wrong place) the message appears; "No vertex found. Click left button to try again. Middle button to stop."

See Figure 3.5.

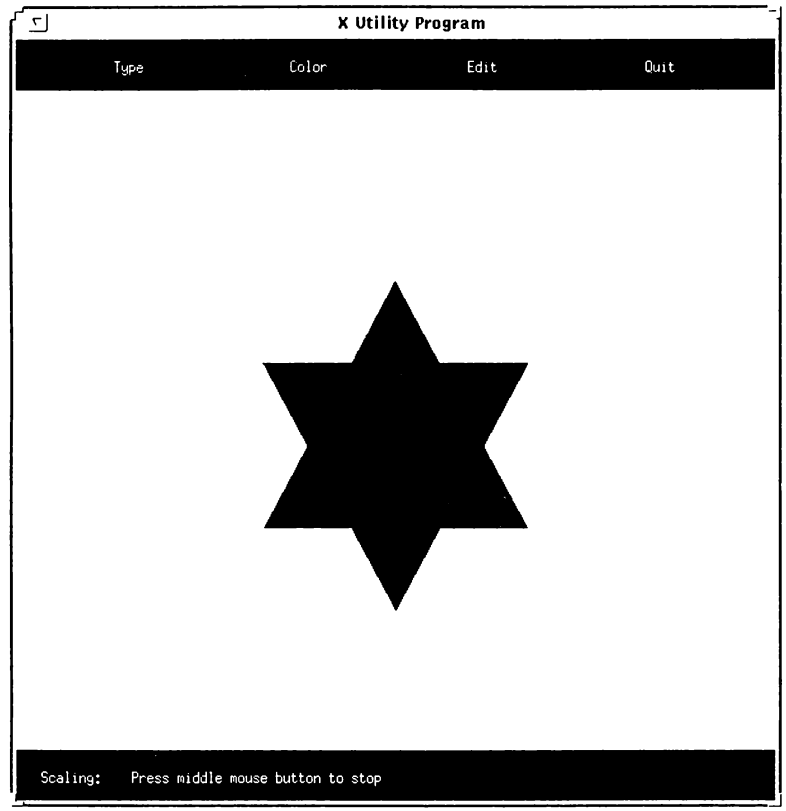

Figure 3.4 Scaling action. Scaling the polygon down in size

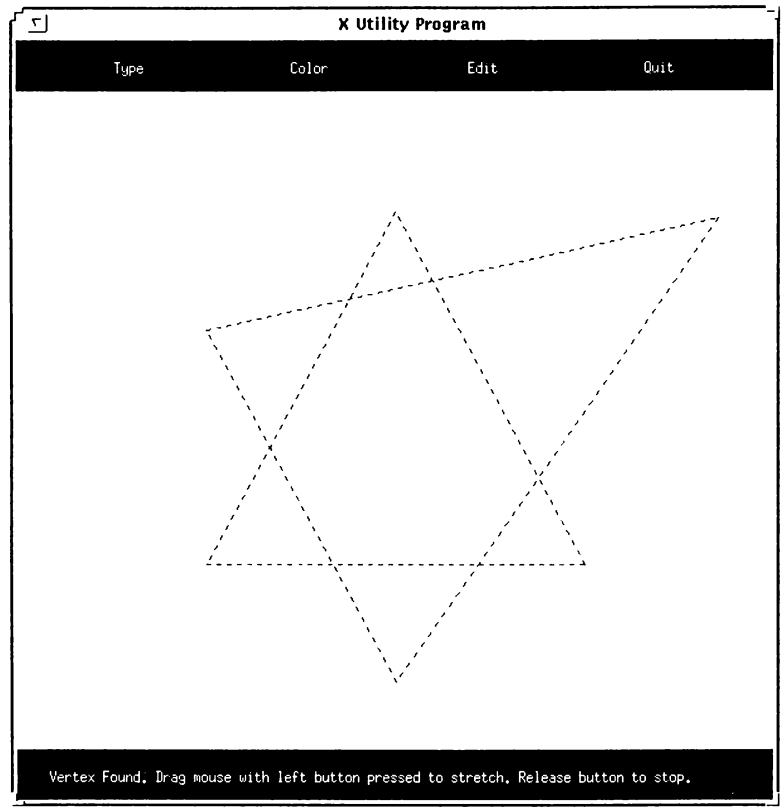

Figure 3.5 The redrawn polygon after rubberbanding has finished

3.d. Reset: Clicking on the Reset button restores the polygon to its default shape and color.

4. Quit: The Quit CascadeButton brings up a pulldown menu with one menu item; Exit. Clicking on this menu button closes the application.

All the above mentioned functionalities are handled interactively. At any time, any of the various options can be selected. Also, all the different branches of the main menu are always accessible and selection process or sequence is kept flexible. For example, a user can decide to turn the polygon into a filled polygon and then decide to change its color. The user can then scale the polygon up and then choose another color. He/she can then rotate it and redraw it in solid lines and choose yet another color.

Let us now examine some of the salient features of the application. While designing the application, the foremost guiding factors have been:

- To build an application that can effectively accommodate principles of user-friendly software.
- To design an application that offers scope to implement the fundamental interaction tasks.
- To build an application which shows the effectiveness and convenience of writing interactive applications using the X Window System

3.4 General Guidelines Used

Some general guidelines have been embodied in the design of this application. This application has been written solely with Xlib, with the purpose of demonstrating that it is possible to develop the "look and feel" of a standard interface, one that is consistent

with the features of the most commonly used toolkits, without the aid of any toolkit. Each element of the interface, like menus and buttons, has been individually built and configured for this application. A very similar interface has been built for the latter application, using a toolkit. This shows the versatility of X in building standardized and uniform user interfaces.

This is a simple system. It is based on the "What you see is what you get" principle. It allows actions to be initiated and controlled by the user. At every stage the user can decide on the course of action. Also, the "see and choose" process has been made as intuitive as possible to impart a degree of naturalness.

The response to user actions is immediate and visible - allowing the user a speedy sense of accomplishment. It also reduces the amount of memorization of command codes.

It never deadens an user. There is at every level, a quit option which allows the user to backtrack or start over.

Communication is visual. There is no need to decompose complex and multiple commands. The load on the user's memory is reduced to practically nothing. Also, all linguistic communication is 'natural' and understandable by any non-computer professionals.

3.5 Selection Techniques Used

This application relies heavily on a direct manipulation technique. The central idea is the visibility of options and actions of interest. The method used is 'select and click.' Rather than elaborate commands, the set of alternative choices are a collection of displayed entities. These entities are simple menus, each with a name given after the very function it serves. An analogy can be drawn here with the cover of a book. A to-be reader

might not know exactly what the book is all about, but can get a fairly good idea just by reading the name of the book. This makes the entire process easy and intuitive.

Command Selection. Each command is visually represented as a menu item. The organization of menus follows a hierarchical system. The primary action groups, designed as labels bring up pulldown menu items in a ordered sequence. The user chooses a phase from the main menu set and then selects sub-phases from subsequently displayed menus. An avenue is always available to the user to backtrack his/her way to the main menu.

Item Order. Menus are logically grouped. The goal has been to display the most general menus at first and then allow branching out off each major type.

3.6 Interaction Techniques Shown

Positioning. A positioning task can be carried out by simply reconfiguring the display window with the polygon in it. The window can be dragged to another location and the contents of the window, which in this case is the polygon, will move with the window and be relocated as the window is repositioned. The dragging and repositioning of the window along with its contents is automatically taken care of by X.

In the same way, the window can also be resized. In that case, the application takes care to proportionately (proportionate to the length/height of the resized window) resize the polygon.

Stretching. The rubberbanding technique gives the user an easy way to try out different line positions, before settling on a new line. Therefore it offers the user a very interactive way to draw a new polygon, furnishing the user a total sense of control.

Manipulate. The scaling feature has been adopted with the primary purpose of demonstrating easy manipulating technique that is interactive and fairly simple.

Shaping. It is quite useful to implement shaping features since they are widely needed and used in all sorts of applications. In this case, the Poly option gives the user an opportunity to bring about an immediate transformation of the shape of the polygon.

All these features which form the rudiments of a comprehensive interface, have been presented to the user in an intuitive manner. The user can merely look at a color menu and make a selection. He/she does not have to remember whether the color he/she wants to select is a shade of light steel blue or antique white.

3.7 A General Break-up of the Program

The basic steps that were taken in this program are as follow:

- Connect the client to the server.
- Create all the windows needed. While doing so, allocate a background color for each window and their size and location.
- Build routines to draw the contents of each window, for example, the polygon or a string inside a window. While doing so, create a graphics context for each window to control the action of drawing requests.
- Select the types of events the application needs to receive.
- Display or map the windows.
- Loop for the desired events.

- Respond to the Expose event resulting from any window being mapped or becoming visible, by calling routines to draw text and graphics.
- Respond to different types of events, for example, ButtonPress or ButtonRelease by calling appropriate function code.

3.8 Programming with Xlib

It might be interesting and helpful to show some fragments of programming code here. For a detailed explanation of all the variable types and the X function calls used, please refer to chapter 2.

The pieces of code shown here are in C language and has adhered to the C syntax and C style of commenting. The names of most of the variables are quite self-explanatory. It can be safely assumed that a reader/programmer not familiar with the C language will not find it very difficult to follow the basic structure and logic of the code.

As is usually the practice, the code is composed of a main program and several subroutines. The main routine calls the subroutines and sets up the event-gathering loop and responds to each event, performing the appropriate action, for example, call another subroutine depending on the type of events and the type of information contained in the event structure.

Windows: windows have been created simply by routines that call the XCreateWindow() function. The layout of the window is determined by passing the coordinates of the upper-left corner of the window and the desired length and height.

The following piece of code shows how a window can be created.

```
/* Definition for the event_mask */
```

```
#define VALUE_MASK CWBackPixel | CWBorderPixel | CWEventMask |
CWOverrideRedirect
#define NEWEVENTS ExposureMask | ButtonPressMask |
ButtonReleaseMask | ButtonMotionMask | KeyPressMask | StructureNotifyMask |
OwnerGrabButtonMask
XSetWindowAttributes xswa;
Window MainWinId;


/* Attributes of the window to be set at creation time should be set in the xswa structure
*/
xswa.event_mask = NEWEVENTS; /* Set of events that should be saved */
xswa.background_pixel = white;
xswa.border_pixel = gold;
MainWinId = XCreateWindow(display, root, 100, 100, 600, 600, 10, CopyFromParent,
InputOutput, CopyFromParent, VALUE_MASK, &xswa);
```

XCreateWindow returns the Id of the created window. VALUE_MASK specifies which window attributes are defined in the xswa structure. This window will have its upper-left origin at 100, 100 and be 500 pixels long and broad. The color of the background will be white and the border is gold.

Color: It is easy to allocate a desired color for any window simply by looking up the color database provided by the X11 lib (/usr/lib/X11/rgb.txt) and selecting the values of Red, Green and Blue, to get the pixel value for that shade. For example, 112-Red, 219-Green and 147-Blue will provide a shade of aquamarine.

The following piece of code shows how to set a color.

```
XColor xc;
```

```
Display *display;
Colormap cmap;
unsigned long gold;


xc.red = 204*255; xc.green = 127*255; xc.blue = 50*255; /* Scaled by 255 */
XAllocColor(display, dcmap, &xc);
gold = xc.pixel; /* Returns in the xc structure the pixel value */
```

Now this pixel value can be passed to the attributes structure (xswa) of a window.

Mapping: All windows are mapped by the XMapWindow() call. It is important to remember that the X server does not automatically preserve the visible contents of a window. Graphics visible in a window will be erased when that window is obscured and then exposed. For this reason, it is important for the application to take care to redraw the contents of a window whenever it is exposed.

Drawing: All drawing routines first create a Graphics Context. Each window here has its own GC. The specified components of the GC are set to the values passed in the valuemask argument while creating the GC. These can be easily changed by the XChangeGC() call and again passing in a valuemask for the altered values.

The following piece of code deals with the graphics part.

```
XGCValues gcv;
GC xgc;


/* gcv structure will provide components for the new GC */
```

```
gcv.background = white;
gcv.line_style = LineSolid;
gcv.foreground = black;
/* The returned GC can be used in subsequent requests, but only on drawables on the
same screen */
xgc = XCreateGC(display, MainWinId, GCForeground | GCBackground | GCLinestyle ,
&gcv);
```

Now for purposes of rubberbanding , we need to change the linestyle to draw dashed lines. This can be accomplished merely by changing that attribute of the GC only. The following code will work to that effect.

```
gcv.line_style = LineOnOffDash;
XChangeGC(display, MainWinId, xgc, GCLineStyle, &gcv);
```

Event Processing: Event processing is central to any X program. A client must select the types of events it wants from each window. The selection can be made by setting the event_mask attribute while creating the window.

Every event is represented by a event structure. The type of event is reported in every event structure. For example, in the case of an Expose event, the Type field would be the symbolic constant representing Expose. The structure contains a number of useful information, all of which are needed for writing event-processing functions.

Event processing is often implemented as an infinite while loop beginning with the event-gathering loop and followed by a switch statement that branches according to the event type received. Within each branch, there are additional branches of subroutine calls.

The pieces of code shown above are obviously not comprehensive. Neither are they complete. It has been shown primarily to offer a flavor of Xlib programming. The author hopes that it would be useful for readers who are attempting to begin programming with X and especially those who are contemplating building interactive applications in X.

Now that we have dealt with and examined an application of a smaller scale, it would be appropriate to go on to our next application which is bigger and requires far more complex programming. The above exercise would serve well as a stepping stone.

CHAPTER 4

The Final Application

The preceding chapters made an effort to illustrate the fundamental aspects of writing interactive applications using the X Window System. The primary thrust of this study has been to explore the basic techniques as well as the nature of X programming in an interactive software setup. This chapter deals with the final part of the study, the final application. This application builds on all the X exercises enumerated in this study, incorporates the fundamental principles of Human Factors as described in the earlier chapters and shows the ease and flexibility with which highly interactive and user-friendly software can be developed in the X environment. The end result is a reaffirmation of the initial premise of this thesis, that X Windows is ideal for building interactive and responsive applications.

For creating the final application, a commercial X toolkit has been used. The toolkit used here is Motif, OSF's (Open Software Foundation) well known and widely used toolkit. The purpose of the X toolkit is to provide a simplified approach to graphical user-interface programming. To simplify development, each of the user-interface elements of a graphical application, like scrollbars, command buttons, dialog boxes, popups or pulldown menus should ideally be available ready-made, so that the programmer need only integrate them with the application code. That is exactly what a toolkit like Motif provides. Motif is a library of ready-to-use user-interface elements. The toolkits can make X programming much easier, with built-in user configurability and

built-in code for interaction with the window manager. Programmers are most often advised to use a toolkit for easy and quick application development.

It is not in the realm of this thesis to delve into a detail discussion and analysis of the Motif Toolkit and the structure and technique of writing Motif-based applications. The final application has made use of the Motif Toolkit merely to point out the usefulness and availability of alternatives ways of building X applications. It is a matter of great convenience to implement ready-to-use interface components and provides ample reasons for choosing X Windows as a primary system for building interfaces.

This chapter discusses the main application. But before doing so, it is only relevant to offer a brief overview of the Motif Toolkit. It would help familiarize the reader with the basic workings of a toolkit and the use of such a toolkit in developing the interfacing part of an application. But an in-depth discussion has not been presented in this context. Interested readers may find many manuals on Motif very useful.

In the earlier programs, outlined in this study, Xlib has been used solely to build all the interface elements. Motif provides most of these elements in the form of widgets. It also allows easy integration of event and action oriented programming.

4.1 Motif

As has been mentioned earlier, applications communicate with the server by means of calls to Xlib, which is the low-level library of C language routines. Xlib calls are translated to protocol requests that are passed either to the local server or to another server across the network.

Toolkits implement a set of user interface features, such as menus or command buttons, (generally referred to as toolkit widgets) and allow applications to manipulate these features using object-oriented programming techniques.

The OSF/Motif Toolkit is based on what is known as Xt, the X Toolkit Intrinsics. Xt provides routines for creating and using widgets and quick and easy access to the lower level of X. The Motif widget system is layered on top of Xt, which in turn is layered on top of the X Window System. The widget set includes menus, dialog boxes, scrollbars and so forth, which all work together, to provide a consistent "look and feel" for an application.

4.2 Motif Widgets

Widgets are the different components of the user interface. Every widget is dynamically allocated and every widget belongs to one class. Each class has a structure that contains operations for that class. For a user of existing widget classes, a widget class is a black box, that has certain fixed features and certain configurable features. Each configurable feature is called a resource and the ability to set these features gives the programmer control over the look and feel of the interface.

Every widget is dynamically allocated. Every widget belongs to one particular class and each class has a structure that contains operations for that class. Widgets are used either individually or in combination. Some widgets display information, others change their display in response to user input and can invoke functions when instructed to do so. Aspects of widgets, such as fonts, colors can be customized. Widgets are grouped into several classes, depending on the function of the widget. Logically, a widget class consists of the procedures and the data associated with all widgets belonging to that class.

4.3 Callback Procedures

Callbacks are one of the key features of the Motif widget set. These allow certain procedures to be called when certain events occur within a widget. These events include mouse-button presses, keyboard selections and cursor movements.

Widgets are designed to let the user control the application. Therefore, widgets have the ability to invoke certain sections of the application code. The application can arrange for a widget to invoke application code by registering a callback function for that particular widget, with Xt. Once the application is running, Xt will call these functions in response to some occurrence in the widget. For example, a PushButton widget usually invokes an application function, when the user clicks on the widget. Thus the widget labeled 'Quit' might invoke the code that checks if data has been saved and if so, exits the application.

To add a callback function to any widget, these steps have to be taken:

- The callback procedure has to be written.
- The callback function has to be added to the appropriate widgets.
- The widgets' callback resources have to be set.

4.4 Resources

Widgets come with their properties or resources. When an instance of a widget of a certain class is created, the instance will have certain fixed features which are common to all instances of that class, and certain characteristics which can be changed from one instance to the next. A user of existing widget classes would be most interested in the

configurable features, since these provide flexibility and control. Each configurable feature is called a resource.

Widget classes can declare variables as named resources of the widget. The application can pass the value of widget resources as arguments to the call to create a widget instance, or can set them after creation. Also, as an application starts up, a part of Xlib called the resource manager can read configuration settings placed in a series of ASCII files and Xt can automatically use this information to configure the widgets in the application.

4.5 Using Motif Widgets in Programs

All Motif applications generally have the same basic structure, as follows:

- Include the standard header file for Motif. Include the public header file for each widget class used in the application.

- Initialize the toolkit. The function call to initialize the toolkit creates an application context, establishes the connection to the display server, loads the resource database and creates a shell widget to serve as the parent of application widgets.

- Create widgets. This requires one call for each widget.

- Register callbacks and event handlers, if any, with Xt.

- Realize the widgets. This function has to be called only once in the entire application, passing it the shell widget for the application. This step actually creates the windows for the widgets and maps them onto the screen.

- Begin the loop for processing events. At this point, Xt takes control of the application and operates the widgets.

4.6 The Bugtracking System

The piece of software built as the final application is a 'Bugtracking System.' It is a system by which software bugs or defects can be recorded and tracked for various software projects. Such a system would be deemed very useful for a software manager, managing multiple projects and who needs to constantly record new bugs, track existing bugs and arrange for the systematic removal of these bugs. In fact, in any software company, a considerable amount of time is assigned to the task of bug removing or debugging, as it is commonly called. Most often the project manager, or the person in charge, would assign certain developer/employee to work on removing the bugs and usually, there would be a specified time period by which the task is expected to be accomplished.

Software bugs are also frequently reported by users and/or customers. Information pertaining to the customers or users reporting the bug is often preserved as part of project development and management. Also, as and when the bugs are removed, the project status needs to reflect the state of the bugs.

The software built here, serves the above purposes. It is a system by which comprehensive information can be recorded about each software bug, as and when they are found. It can be used by managers, engineers/programmers as well as technical support staff. The records can be checked at any time by querying the system. Queries can be made about the bugs in a flexible manner, using various information categories, allowing the user a wide choice of options to query by. Updates or changes are allowed

for project management. The application is written in C language and has built all the elements of the graphical user interface with X and Motif.

4.6.1 The Design

This application is designed around three major categories of functions. These are:

- Entering or recording new bugs
- Querying existing bugs
- Editing or updating current bugs

The primary interfacing medium has been implemented through a form-filled layout of screen design. The interaction windows consist of all the elements of data display and data entry. These data entry and data display forms contain all the necessary data entry or display fields, buttons, list boxes etc.. Users interact with the system primarily through these forms.

The main window consists of a menubar with two menu choice labels. These are:

- Bug
- Quit

The Quit option lets the user exit the application.

The Bug menu leads to the different operations available.

Selecting the Bug menu brings up a pulldown with the two major groups of actions as menu items. These are:

- New
- Query

New. New is the means by which new bugs are recorded. New brings up a entry form with several fields for entering different categories of data and marking relevant checkpoints The user is given the choice to enter and mark the following categories of data for each bug record.

Bug Id. Each bug is to be assigned an index or id for the purpose of convenient tracking. The New menu item brings up a set of submenus, that offer two methods of assigning ids. These are, Automatic Id and Manual Id. If the Automatic option is selected, the system generates a sequential index and allocates that index for the new record to be entered. Whereas, if the manual option is chosen, the user is prompted through a dialog box, to assign his/her own id for that particular record. Once the user has entered a new bug id, the system verifies that the given id is not in use already, and accordingly allocates the id or prompts the user for another id.

Once a manual id is chosen, the entry form displays the index in the bug id field and sets it to a non-editable mode.

Platform. The names of platforms in use for current projects, for example, IBM, DEC etc., is displayed through a list box and the user can make a selection. The selected platform name then becomes part of the bug record.

Date. An entry field is offered for the user to mark the current date for the bug record.

Customer. More often than not, bugs are reported by customers and requests are made by them to fix these problems. Customer information is an important part of tracking any bug removing scheme. Frequently, the customers are provided with updates of bug fixing.

Therefore, customer name, for example, the name of the companies, are useful part of a bug record.

Contact name. Most communication with a client company is done through a certain staff member of that company. The name of this contact can be kept as a part of the customer information.

Address. The user can enter the address of the customer.

Telephone, Fax and Email. The user can enter a telephone number, a fax number and a email address as part of customer information.

Description. For a manager, keeping track of several different projects, a short description of a bug could be very useful. Therefore, the user is allowed an entry of a scrollable and short description of individual bugs.

Keyword. The user can enter a keyword, for example, the nature of the bug like lighting bug, for the purposes of quick review.

Assigned to. The name of the person assigned to work on the particular bug can be recorded.

Expected time to fix. The user is expected to enter the number of days estimated to be needed for removing the bug.

Date assigned. This is the date on which, a person is actually assigned to start working on the bug.

Expected fix date. This is the due date, by which the bug ought to be removed.

Priority. The user can select a category of priority from a list box (a range of 1 to 5) for his own convenience.

Status. The user can select two possible status options for a bug. Open or Closed.

The user is also provided with a Record Status button which toggles between a locked and a unlocked mode. When entering data through the New menu option, this button is set to the unlocked mode.

Save. Once the user has completed entering all necessary data, he/she is to click on the Save button for saving all the data for the new record created. The record status mode is then switched to locked.

Cancel. The Cancel button cancels the entry function, discards all entered information.

See Figure 4.1.

Query. If the query mode is selected from the pulldown menu, a query window appears in a similar form-filled layout. The user can enter or mark checkpoints for one or multiple data fields, to be used as categories to query by. The available categories are as following.

- Platform
- Priority
- Date, greater than or equal to
- Date, less than or equal to
- Customer
- Contact Name
- Keyword
- Assigned To

- Status

- Query All (Lists all existing bugs)

See Figure 4.2.

After entering data for one or several of these fields, the user can select a button to start query or select the Cancel button to abort the query.

The result of a query is displayed in a display form similar to the entry form with data fields and checkpoints reflecting the state of the fetched record or records.

The records are in locked mode by default, when a fetched record is being displayed.

The display form also contains some additional buttons. These are:

Prev: The Prev Back button the user to view the previous fetched record, in case the query fetches more than one record.

Next. The Next button allows the user to view the next fetched record, in case the query fetches more than one record. This button is grayed out, if that is not the case.

Save. The Save button is significant for the purposes of editing bug records.

Edit. The Save button in the display form is provided with the intention of allowing quick and interactive editing. The user can update or change record data while viewing them and then press Save to make the changes permanent. To edit records displayed after a querying operation, the record status has to be manually switched to a unlocked mode, before any editing can be done. It is important to note the fact that only one person can edit a record at one time. A record will automatically get locked when a user is updating it.

The software described above, can take full advantage of the multi-user and network

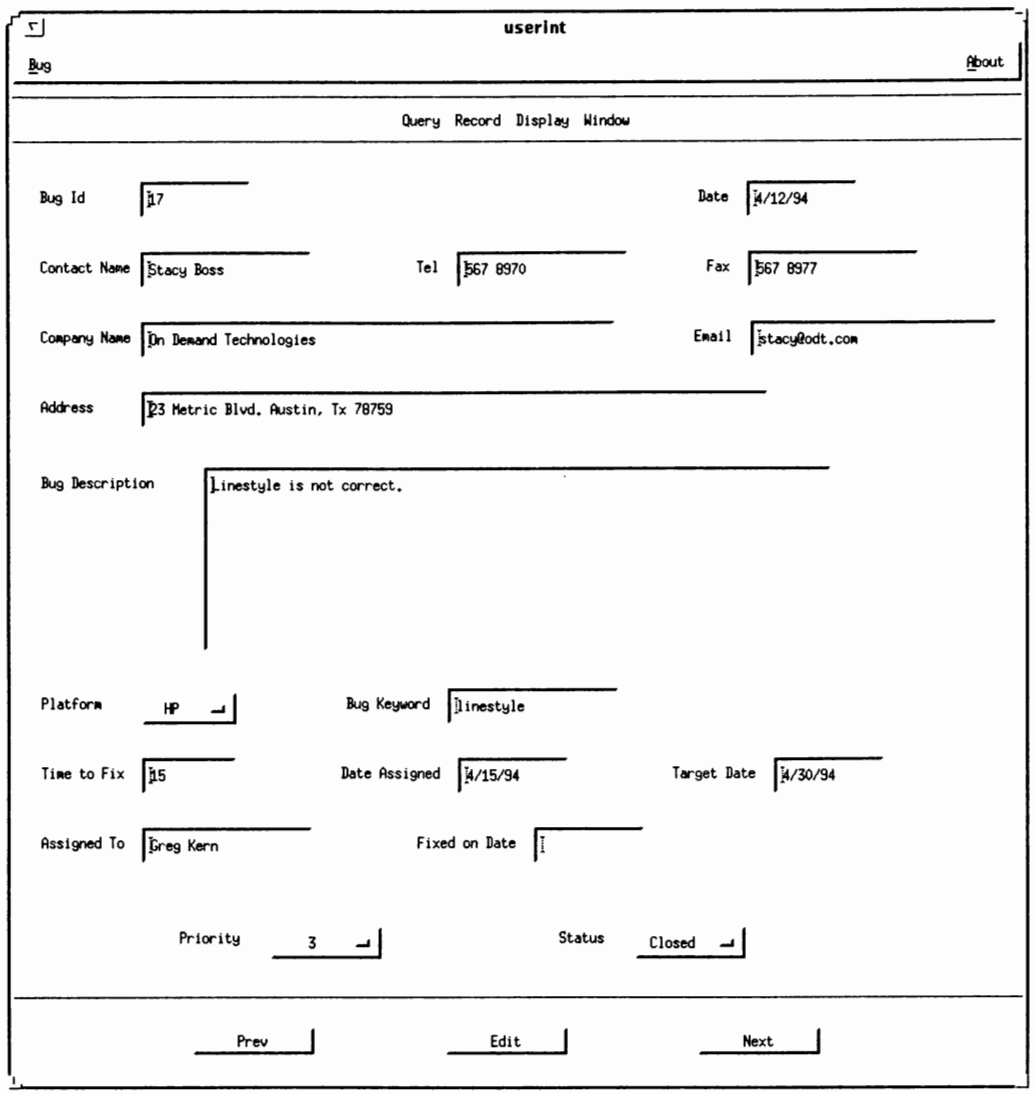

Figure 4.1 The window for data entry for a new bug record

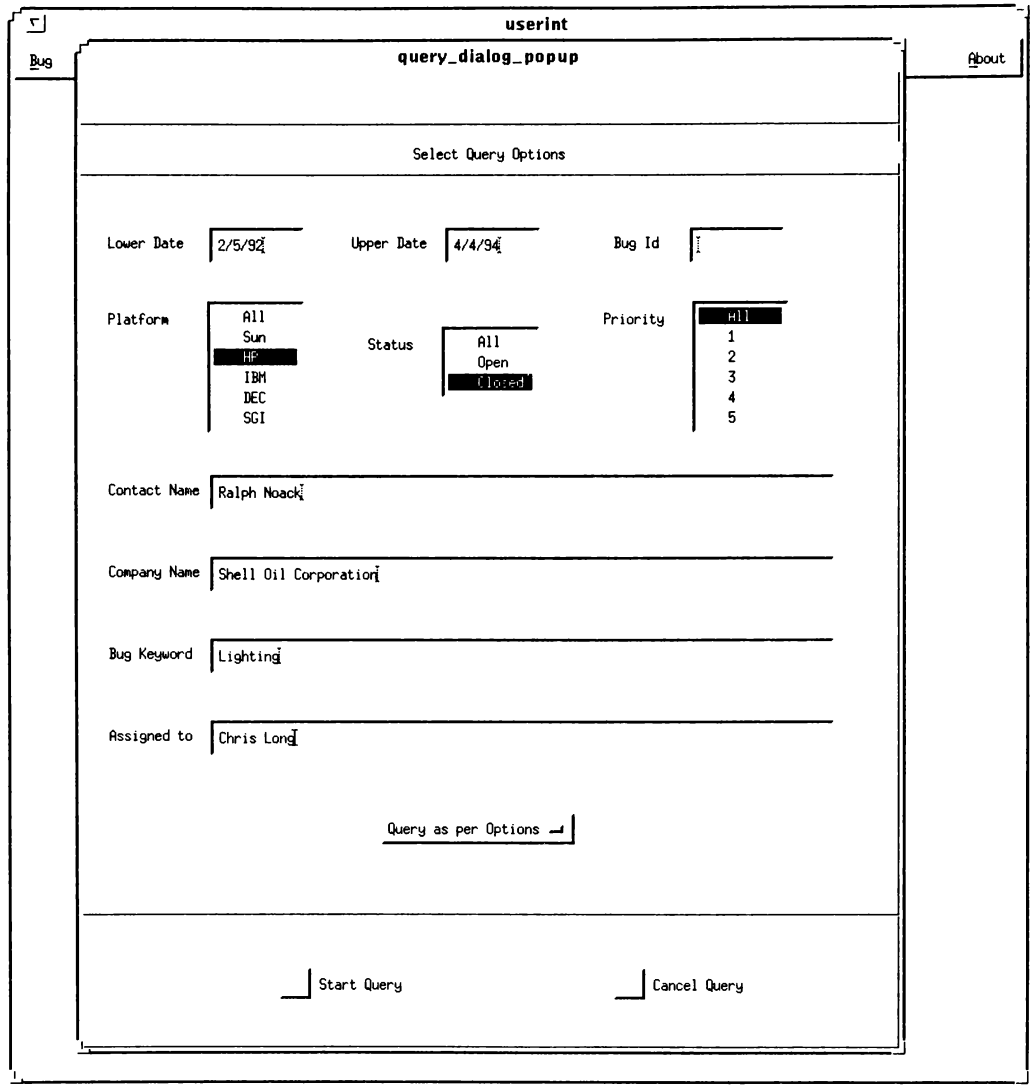

Figure 4.2 The window for entering data to query by

based nature of X. It can be accessed by many users remotely, over a network. Projects in different sites can be possibly monitored simultaneously. For any, software developing environment, this could be an extremely efficient project management tool.

CHAPTER 5

Conclusion

The primary focus of this study has been to delineate the effectiveness of the X Window System as a platform for developing interactive applications as well as to distinguish the essential elements of interfacing mechanisms that go into the making of user-friendly applications. Readers have been led through a brief session of the X Window System and the case studies have exemplified integration of the fundamental principles of computer-human interface design in an X environment.

At this point, certain observations can be safely outlined and certain conclusions that support the proposition laid down by this study, can be arrived at.

The X Window system can be upheld as a highly versatile system in the sense that it supports a wide range of interaction techniques as well as allows creation and use of a multitude of graphical interface components. Moreover, the existence of various toolkits can make the building of the interface quick and convenient. At the same time, the 'Free Policy' of X gives the designer total control and flexibility.

Events imply a philosophy that the program should respond to the user's actions, not the other way around. The program must be able to listen to several types of events at once and jump back and forth when acting on them. X provides the key elements that interact in the design of a user interface, such as the hierarchy of windows and the selection and processing of events, chiefly pointer and keyboard events. Since these

device events propagate through the hierarchy depending on whether they are selected for each window, for every user action, it is possible to provide a path, through event-handling code, that yields some sort of response to the user. This is surely of vital importance for a good interface design.

X provides strong graphics capability, a invaluable resource for developing graphical user interface based applications. The X resource Graphics Context (GC) facilitates the optimized use of graphics primitives, since the appearance of everything that is drawn within a window can be conveniently controlled by the GC that is specified with each graphics request. This provides the designer enormous flexibility.

X also provides ample drawing primitives, which are easy-to-use routines capable of drawing lines, points, rectangles, circles, etc..

A wide range of colors is available through X. A typical X application allows the user to specify colors for the background, border of the windows, color for the cursors and foreground and background colors for drawing text and graphics. X also supports a wide variety of systems with different screen hardware.

The window manager provides easy control over the layout of windows on the screen, responding to user requests to move, resize, lower or iconify windows. The window manager may also enforce a policy for window layout. X provides features and routines to give the window manager the authority they need to control window layout and flexibility to provide a good interface.

Last, but not the least, X is highly portable across platforms and even operating systems. Therefore, applications in X can be ported with ease and user interfaces can remain unchanged across platforms, without affecting high performance. X provides overall comprehensive controls and methods for implementing the most useful features of

user-interface management and thus, can be undoubtedly hailed as a primary choice for building interactive applications.

BIBLIOGRAPHY

Adrian, Nye. X Protocol Reference Manual. California: O'Reilly and Associates Inc., 1990.

Adrian, Nye. X Toolkit Intrinsics Programming Manual. California: O'Reilly and Associates Inc., 1990.

Adrian, Nye. X Toolkit Intrinsics Reference Manual. California: O'Reilly and Associates Inc., 1991.

Adrian, Nye. Xlib Programming Manual. California: O'Reilly and Associates Inc., 1990.

Adrian, Nye. Xlib Reference Manual. California: O'Reilly and Associates Inc., 1990.

Brown, Marlin C. Human-Computer Interface Design Guidelines. New Jersey: Ablex Publishing Corporation, 1988.

Kasik, David. Controlling User Interaction. Computer Graphics, Volume 10. New York: Association for Computing Machinery, 1976.

Foley, James and Chang, P. Human Factors of Computer Graphics Interaction Techniques. IEEE Computer Graphics and Applications. California: IEEE Computer Society, January 1984.

Foley, James, Andries, van Dam, Feiner, S., Hughes, J. Computer Graphics, Principles and Practice. Massachusetts: Addison-Wesley Publishing Company, 1990.

Heckel, Paul. The Elements of Friendly Software Design. New York: Warner Books, 1984.

Marcus, Aaron. Designing the Face of an Interface. IEEE Computer Graphics and Applications. California: IEEE Computer Society, January 1982.

McCormick, Ernest J. and Sanders, Mark. Human Factors in Engineering and Design. New York: McGraw-Hill, 1987.

McMinds, Donald L. Mastering OSF/Motif Widgets. Massachusetts: Addison-Wesley Publishing Company, 1992.

Monk, Andrew, ed. Fundamentals of Human-Computer Interaction. Florida: Academic Press, 1984.

Open Software Foundation. Programmer's Guide. New Jersey: Prentice Hall, 1991.

Rubinstein, Richard and Hersh, Harry. The Human Factors: Designing Computer Systems for People. Massachusetts: Digital Press, 1984.

Shackel, B., ed. Man-Computer Interaction: Human Factors Aspects of Computers and People. Maryland: Sijthoff and Noordhoff, 1981.

Shneiderman, Ben and Badre, Albert. Directions in Human/Computer Interaction. New Jersey: Ablex Publishing Corporation, 1982.

Shneiderman, Ben. Designing the User Interface: Strategies for Effective Human-Computer Interaction. Massachusetts: Addison-Wesley Publishing Company, 1987.

Shneiderman, Ben. Software Psychology: Human Factors in Computer and Information Systems. Massachusetts: Wintrop Publishers, 1980.

Vassiliou,Yannis, ed. Human Factors and Interactive Computer Systems. New Jersey: Ablex Publishing Corporation, 1984.

Wixon, D. and Whitehead, J. Building a User Defined Interface. Communications of the ACM. New York: Association for Computing Machinery, October 1983.

Young, Douglas and Pew, John. The X Window System. Programming and Applications with Xt. New Jersey: Prentice-Hall Inc., 1992.