UNIT 1 GRAPHICAL USER INTERFACE

Structure **Page Nos.**

1.0 INTRODUCTION

The **Graphical User Interface (GUI)** is one of the most revolutionary changes to occur in the evolution of the modern computing system. The interaction between man and computer in the design of operating system has changed from a character-oriented system to the now more **graphics-oriented** system. This revolution has increased the accessibility and usability of computer systems to the general public.

We will look at GUI features of the various operating systems with little knowledge of operating system or memorized commands. The software providing such features supports **Modern User Interface** concept such as desktop metaphor, which makes computers available to the majority of people who are either novices or non-programmers. The personal computer was invented for these users.

We will start with the basic definition move on to basic components and finally look into some systems to find out what GUI facilities they are supporting.

1.1 OBJECTIVES

After going through this unit, you should be able to:

- define what GUI is and how it is different from character oriented system,
- define all the terms related with GUI, and
- identify important features of several GUIs.

1.2 WHAT IS GRAPHICAL USER INTERFACE?

The term "user interface" originated in the engineering environment in the late 1970s. Virtually everyone who interacted directly with computers had been an engineer and a programmer, but new kinds of users were emerging: the non-programming user. These users often reacted more negatively to difficulties in dealing with a machine. New forms of interaction were needed, new interfaces, were required, attention flowed to "the user interface".

With the introduction of the Macintosh in 1984, Apple Computer popularised the user interface as it is known today. Apple's user interface is now commonly referred to as a Graphical User Interface or GUI. The GUI has become associated with a common feature set available in a number of product offerings. Common features include:

- Secondary user-input devices. Usually a pointing device and typically a **mouse**.
- Point and shoot functionality with **screen menus** that **appear** or **disappear** under pointing-device-control.
- **Icons** that represent files, directories and other application and system entities.
- **Dialog boxes, button, sliders, check boxes** and many other **graphical metaphors** that let the programmer and user tell the computer what to do and how to do it.

Today's GUIs have expanded basic functionalities to support not only graphics, but also dimensions, color, height, video, and highly dynamic interaction. Modern user interfaces can simulate a very realistic view of a real, three-dimensional world.

1.3 EVOLUTION OF HUMAN AND MACHINE INTERACTION

The primary means of communication with computers earlier had been through command-based interfaces. In command interfaces, users have to learn a large set of commands to get their job(s) done. In earlier computer systems paper tapes, cards and batch jobs were the primary means of communicating these commands to the computers. Later, time-sharing systems allowed the use of CRT terminals to interact/communicate with the computer. These early systems were heavily burdened by users trying to share precious computer resources such as CPU and peripherals.

The batch systems and time-sharing led to command-driven user interfaces. Users had to memorise commands and options or consult a large set of user manuals. The early mainframe and minicomputer systems required a large set of instruction manuals on how to use the system. In some systems, meaningful terms were used for command names to help the end-user. But in other systems the end-user had to memorise several sequences of keystrokes to accomplish certain tasks.

Early users of computes were engineers and what we now call expert users; users who had a lot of interest in knowing more about computer systems and the technology. Command line interfaces were acceptable to the majority of these users. In the 1970s, computers were introduced to a new class of users: secretaries, managers and non-technical people. These new users were less interested in learning computer technology and more interested in getting their jobs done through the machine. The command-based interfaces caused many of these new users to develop computer phobia. Imagine the thought of memorising commands made up of "Control-Alt-Del" to boot the system.

To make life easier for the end –user, a large collection of devices have been invented to control, monitor and display information. The early (and still widely used) peripherals are the keyboard and the video terminal. But, it was not until the late 70s, that research projects at some universities led to the invention of pointing devices and windowing systems. The mouse and joystick were among some of the few pointing devices that were invented in this period. Also, research pioneers invented the notion of splitting the screen to allow multiple windows and direct manipulation of objects.

In the 70s, researchers designed powerful new workstations armed with graphical user-interfaces. The basic assumption of these new workstations was that one user could have a powerful desktop computer totally dedicated to that user's task. Thus, the computer is not only used to perform the task, but can also provide a much more intuitive and easy-to-use environment. In this unit we will examine the common GUIs.

1.4 COMMON GRAPHICAL USER INTERFACES

This section presents a list of terms used commonly with the graphical user interface (GUI). GUIs are systems that allow creation and manipulations of user interfaces employing windows, menus, icons, dialog boxes-mouse and keyboard. Macintosh toolbox, Microsoft Windows and X-Windows are some examples of GUIs.

1) Pointing Devices

Pointing devices allow users to point at different parts of the screen. Pointing devices can be used to invoke a command from a list of commands presented in a menu. They can also be used to manipulate objects on the screen by:

- Selecting objects on the screen
- Moving objects around the screen, or
- Merging several objects into another object.

Since the 1960s, a diverse set of tools have been used as pointing devices including the light pen, joystick, touch sensitive screen and a mouse. The popularity of the mouse is due to the optimal coordination of hand and easier tracking of the cursor on the screen.

2) Pointer

A symbol that appears on the display screen and that you move to select objects and commands. Usually the pointer appears as a small angled arrow.

3) Bit-Mapped Displays

As memory chips get denser and cheaper, bit displays are replacing character-based display screens. Bit-mapped display made up of tiny dots (pixels) are independently addressable and much finer resolution than character displays. Bit-mapped displays have advantages over character displays. One of the major advantages is graphic manipulation capabilities for vector and raster graphics, which presents information in the final form on paper (also called WYSIWYG: What You See Is What You Get).

4) Windows

When a screen is split into several independent regions, each one is called a window. Several applications can display results simultaneously in different windows. *Figure 1* presents a screen with two windows.

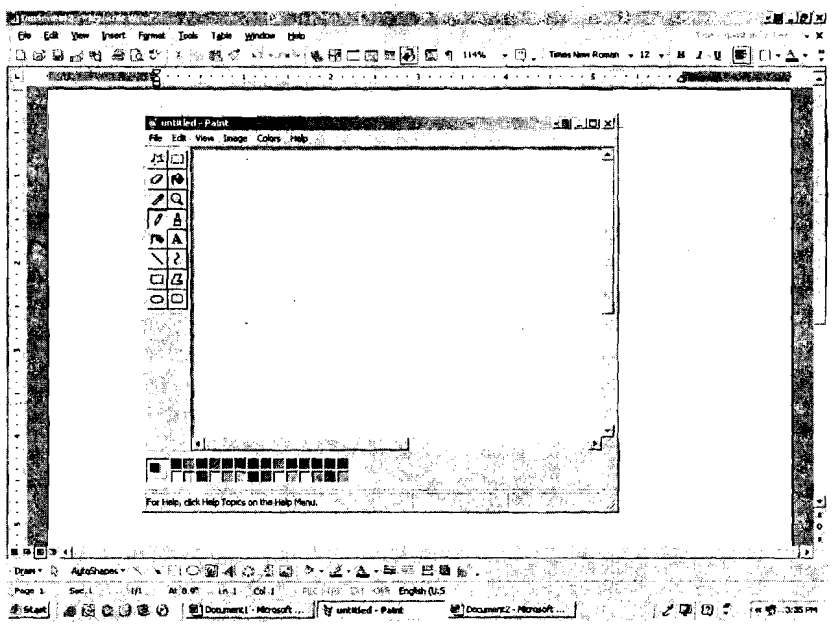

Figure 1: Screen with two windows

The end-user can switch from one application to another or share data between applications. Windowing systems have capabilities to display windows either **tiled or over-lapped,** *Figure 2.* Users can organise the screen by resizing the window or moving related windows closer.

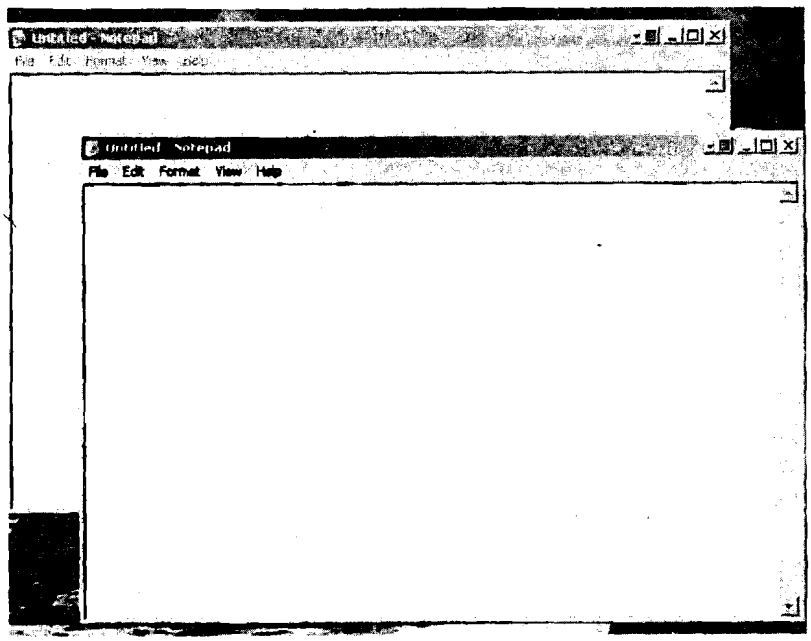

Figure 2: Overlapped Windows

5) Menus

A menu displays a list of commands available within an application (*Figure 3*). From this menu, the end-user can select operations such as File, Edit or Search. Instead of remembering commands at each stage, a menu can be used to provide a list of items. Each menu item can be either a word or an icon representing a command or a function. A menu item can be invoked by moving the cursor on the menu item and selecting the item by clicking the mouse.

Instead of memorising commands to each stage, the user selects a command from a menu bar displaying a list of available commands. For example, *Figure 3* displays the menu bar. This menu bar displays a list of commands available such as File, Edit and the appropriate action is taken.

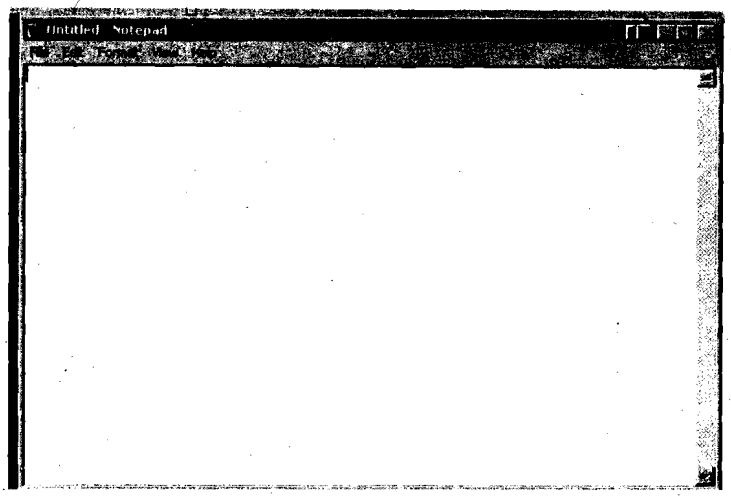

Figure 3: Menu Bar

When a menu item is invoked it could cause other menus, called **pull-down menus**, to appear. **Pull-down menus** (*Figure 4*) are used to present a group of related commands or options for a menu item. *Figure 4* presents the File pull-down menus.

Pull-down and **pop-up** menus display option commands available for each selection. *Figure 4* shows the pull-down menu displayed when the Character menu item is selected. The user can then select from different character styles.

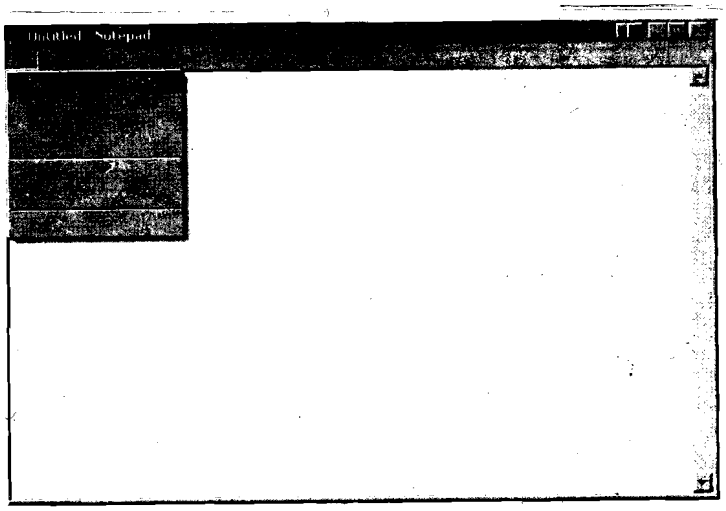

Figure 4: Pull-down Menu

6) Dialog boxes

Dialog boxes (*Figure 5*) allow more complex interaction between the user and the computer. Dialog boxes employ a collection of control objects such as dials, buttons, scroll bars and editable boxes. For example, in *Figure 5*, a dialog box is used to open a file.

In graphical user-interfaces, textual data is not only a form of interaction. Icons represent concepts such as file folders, wastebaskets, and printers. Icons symbolize words and concepts commonly applied in different situations. *Figure 4* shows paint utility with its palette composed of icons. Each one of these icons represents a certain type of painting behaviour. Once the pencil icon is clicked, for example, the cursor can behave as a pencil to draw lines. Applications of icons to the user-interface design are still being explored in new computer systems and software such as the NeXT computer user interface.

Dialog boxes are primarily used to collect information from the user or to present information to the user.

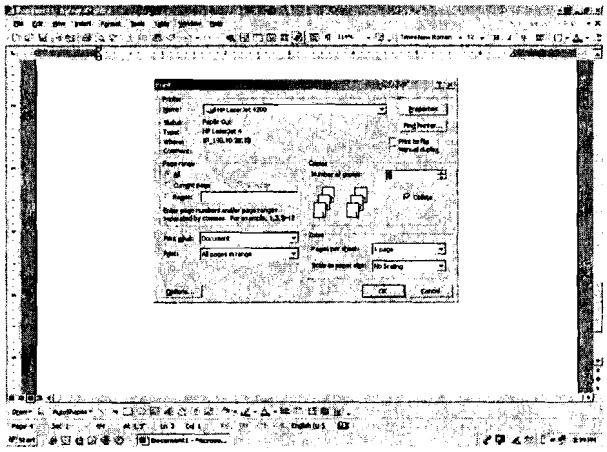

Figure 5: Dialog Box

Among the information obtained are the number of copies and page numbers to be printed. Dialog boxes are also used to indicate error message in the form of alert boxes. Dialog boxes use a wide range of screen control elements to communicate with the user.

7) Icons

Icons are used to provide a symbolic representation of any system/user-defined object such as file, folder, address, book, applications and so on. Different types of objects are represented by a specific type of icon. In some GUIs, documents representing folders are represented by a folder icon (*Figure 6)*. A folder icon contains a group of files or other folder icons. Double clicking on the folder icon causes a window to be opened **displaying a list of icons and folder icons representing the folder's contents.**

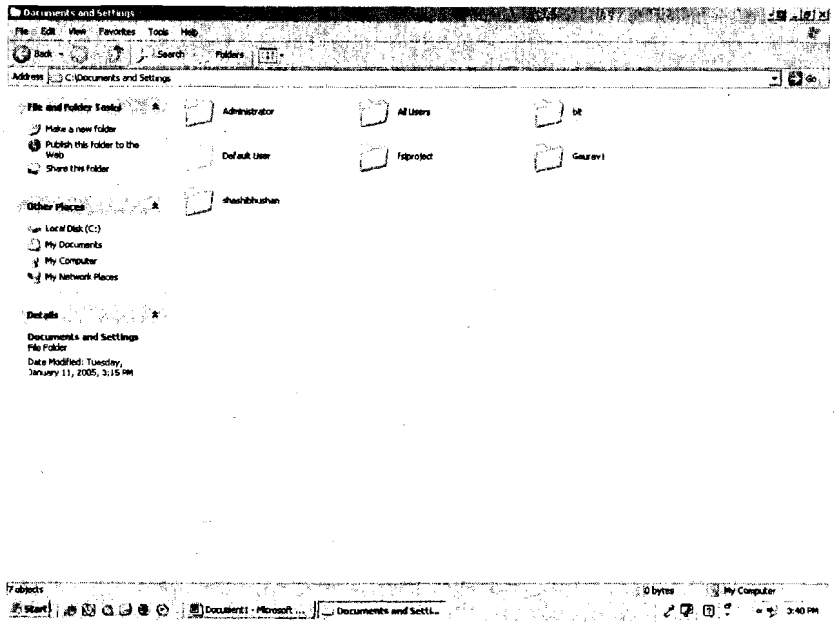

Figure 6: Icons

8) Desktop Metaphor

The idea of metaphors has brought the computer closer to the natural environment of the end-user. The concept of physical metaphor paradigm, developed by Alan Kay, initiated most of the research for graphic user interfaces based on a new programming

approach called object-oriented programming. Discussion of this subject is beyond this unit. The physical metaphor is a way of saying that the visual displays of a computer system should present the images of real physical objects.

For example, the wastepaper basket icon can be used to discard objects from the system by simply dragging the unwanted objects into the dustbin, as in real life. The desktop metaphor probably has been the most famous paradigm. Because of the large set of potential office users, this metaphor can have the most dramatic effect. In this paradigm, the computer presents information and objects as they would appear and behave in an office, using icons for folders, in-baskets, out-baskets and calendars.

In a desktop metaphor, users are not aware of applications. Users deal with files, folders, drawers, a clipboard and an outbox. Instead of starting the word process and loading file, users merely open the report document, which implicitly invokes the word processor. Clicking the mouse on an icon representing the report causes the word processor to get started and to load the report file implicitly. Today, several computing environments provide this capability.

9) The 3D GUI

The desktop metaphor GUI is 2½ D. It is 2D because its visual elements are two-dimensional: they lie in the xy plane, are defined in 2D coordinates, are flat and contain only planar regions (areas). It is 2½ D because where visual elements overlap they obscure each other according to their priority. In a 3D GUI the visual elements are genuinely three-dimensional: they are situated in xyz space, are defined in terms of 3D coordinates, need not be flat and may contain spatial regions (volumes).

The design considerations for a 3D GUI appear more complex than for a 2½ D GUI. To begin with, the issues of metaphor and elements arise afresh. The desktop metaphor with its windows, icons, menus and pointing device elements is firmly established for 2½D GUIs. In contrast no clearly defined metaphor and set of elements for 3D GUIs are manifest — yet. 3D GUIs offer considerably more scope for metaphors than 2½D GUIs; there are many metaphors which could be based on our physical 3D environment, including the obvious extension of the desktop metaphor into a 3D environment, including the obvious extension of the desktop metaphor into a 3D office metaphor. On the other hand, much more abstract metaphors are possible, such as one based "starmaps" where objects are simply placed somewhere in "cyberspace". Likewise the elements of a 3D GUI may resemble, or differ substantially from, the elements of the 2½ D GUI.

The various prototypes have been developed to design the same elements in the 3D GUI as in the 2 $^{1/2}$D desktop GUI: windows, icons, menus, a general space in which to arrange the visual elements, a cursor and an input device to manipulate the cursor.

1.5 FUNCTIONALITY OF GRAPHICAL USER INTERFACES

The development environment for most GUIs consists of four major components:

- A windowing system,
- An imaging model,
- An application program interface (API), and
- A set of tools and frameworks for creating interfaces and developing integrated applications.

Windowing systems allow programs to display multiple applications at the same time. Windowing systems include programming tools for building movable and resizeable windows, menus, dialog boxes and other items on the display. Some GUIs contain proprietary windowing systems, such as Macintosh. Others use common windowing systems such as X-window or simple X.

An **imaging model** defines how fonts and graphics are created on the screen. Imaging models handle, for example, typeface and size in a word processor and lines in a drawing program. This component of the system environment has taken on increasing sophistication as applications incorporate complex curves, color, shading and dimension. Some GUIs support more than one imaging model.

The **API** is a set of programming language functions that allow the programmer to specify how the actual application will control the menus, scroll bars and icons that appear on the screen. Like windowing models, APIs align with particular GUIs.

Finally, **GUI development environments** can include toolkits and frameworks. Most of these toolkits are based on object-oriented approach.

Although the structure of the basic development for most GUIs is similar, there are major differences in how the GUI is integrated with the operating system. Some, like the Macintosh and NeXT GUIs, are closely integrated with the operating system. Others, like X Window or Microsoft's Windows, can be set up as options to be selected by the user when the computer boots up.

Programming of software for GUIs across these components is fairly complex and challenging. Commercial developers who want to support multiple environments find their task further complicated by the absence of standards across heterogeneous computing platforms. The higher-level toolkit component is intended to mitigate much of this difficulty.

Although the graphical user interface has become a standard component of most systems, no standards in windowing systems, imaging models, APIs, or high-level toolkits have emerged. However, three major camps of GUIs dominate. The first camp is IBM's System Application Architecture (SAA), which includes primarily Microsoft's Windows and PM (Presentation Manager). The second camp is UNIX systems, usually built around X Window. The third camp is the Macintosh. In the next section we will describe the object-oriented functionality of representative GUIs in these camps, including Windows, X (upon which most of the UNIX camp is based), NeXT and Macintosh.

☞ Check Your Progress 1

1) What is GUI and what are its features?

.....

.....

.....

2) Define the features of the following:

   a) Window

   b) Pull-down menu

   c) Dialog box

   d) Pointing devices

3) What is the difference between Bitmapped and character based displays?

.....

.....

.....

1.6 GUI DESIGN CONSIDERATION: PSYCHOLOGICAL FACTORS

There are some empirical studies that have identified basic psychological factors that one should consider in the design of a good GUI. In this section we focus our discussion to three primary contributing human factors, which are: the physical limits of visual acuity, the limits of absolute memory, and the Gestalt Principle.

Visual Acuity

Visual acuity is the ability of the eye to resolve detail. This refers to the amount of information one can put on the screen. The retina of the eye can only focus on a very small portion of a computer screen, or anything for that matter, at any one time. This is because, at a distance greater than 2.5 degrees from the point of fixation, visual acuity decreases by half. Therefore, a circle of radius 2.5 degrees around the point of fixation is what the user can see clearly.

At a normal viewing distance of 19 inches, 5 degrees translates into about 1.7 inches. Assuming a standard screen format, 1.7 inches is an area about 14 characters wide and about 7 lines high. This is the amount of information that a user can take in at any one time, and it limits the effective size of icons, menus, dialog boxes, etc. If the user must constantly move his eyes across the screen to clearly focus, the GUI design is causing a lot of unnecessary and tiring eye movement.

Information Limits

Once the user has a desired fixation point, there is a limit to the amount of information that the person can process at one time. A GUI design rule of thumb is that the range of options or choices should never be more than five or six. It has been shown that absolute identification using one-dimensional criteria was about seven items, plus or minus two. Miller introduced the concept of recoding as a method that people use to store information. It has also been pointed out that by expanding the identification criteria from one to more dimensions people could handle more choices and remember more. Later researchers have expanded this work to develop the concept that people chuck information together in order to remember more information. This research has a direct impact on GUI design, especially concerning the number of menu items and icons.

Gestalt Principle

The Gestalt Principle states that people use a top-down approach to organising data. This principle can influence how one should organise graphical information on the screen. The Gestalt school of GUI designers have attempted to identify criteria that cause people to group certain items together in a display. Proper grouping results in a necessary redundancy of selection information that aids the user. For example, if the user knows where one item in a group is on a screen, he will expect other like items to be there also. If one groups the items in line with this expectation, it allows for accurate locating and better transfer of information to the user.

The top-down approach also allows for the development of emergent features. An emergent feature is a global property of a set that is not evident when one views each

item locally. Since global processing tends to be automatic, one can argue that an emerged feature reduces the attention demand as a user operates a multi-element display. For this performance enhancement, one must use the Gestalt Principle in the initial placement, and the resulting organisation must be compatible with the user's cognitive view of the task.

1.7 GUI DESIGN CONSIDERATION: STANDARDS

Considering the above psychological factors, one could come to the conclusion that one could easily extrapolate these factors to the design of a good GUI. Empirical studies of GUI show that this intuition is not always the case. It directly leads to the conclusion that a good GUI would use a lot of icons. Unfortunately, too many randomly placed icons violate the limits of absolute memory. Using the Gestalt Principle, one can group like items together using factors like color to add more informational dimensions. Too many colors, however, destroy the global visual grouping of the items. The user then begins to concentrate on the GUI. Any primary cognitive task attention devoted to the interface may interfere with the primary task. One can derive basic GUI standards from basic human factors, however. The standards are the presentation of information, the grouping of information, and information sequencing.

Amount of Information Presented

The amount of information to present is the most basic of GUI design considerations. H.E. Dunsmore showed that making screens less crowded improves screen clarity and readability. As such, GUI designers usually follow the guidance that the interface should display only what the user needs to perform the current operation. Empirical researchers show that limiting the information to that necessary for the user reduces errors and time to perform tasks. Errors and performance time increase as the GUI presents more information. Of course, it requires a thorough analysis of the tasks that the user must perform in order to display only the necessary amount of information.

Compared to a randomly placed screen, a well-designed screen can reduce the time needed to perform a task by as much as 40%. Ways to conserve screen space are:

1) Appropriate use of abbreviations: Many design documents recommend using complete words whenever possible. Due to screen sizing constraints, it is not always possible to use complete words. When complete words are not possible, abbreviations should be contextual and consistent. A good contextual example is "h," which is usually a good abbreviation to use for help. The number of abbreviations should not only be contextual but also be kept to the minimum. As a poor example, in the UNIX system, the "Is" command lists files in a directory. The "Is" command has 17 different one-letter abbreviations that change the output options of the "Is" command. The one-letter abbreviations have little contextual link to the options they represent. In fact, the UNIX system is a good example of what not to do.

2) Avoid unnecessary detail: For example, use whole numbers if one does not need decimals. Keep the window and icon designs clear and simple. Even when users prefer more complex icons, elaborate icons add nothing to performance. Studies show that when icon designs are too complex, time to complete a task actually increases. In studies with 3-D and 2-D graphical displays, users preferred the 3-D displays. There were no differences in performance between the two graphical displays, however.

3) Use concise wording: Screens have limited space. Screen designers should avoid the tendency to place additional data on the screen just because the data is available. More objective limits of screen density vary from the thresholds of

25% to 80%. There is no empirical research that substantiates any performance enhancement with any specific threshold.

4) Use familiar data formats: With more familiar formats, the user will need less information to complete the task. An example for data entry is the standard USA address format of street, city, state, and zip code. In addition to requiring less instruction, the user will perform the operation faster than if the format is unfamiliar.

5) Use tabular formats with column headings: Tabular formats allow for efficient labeling of related data. It is especially preferable for data location tasks. Simply splitting items on one long line into two lines result in productivity improvements of 20%.

Grouping of Information

Given a set of information to display, there are many ways one can display the information. Proper grouping improves the information's readability and can highlight relationships between the information.

There are several techniques to aid in the grouping of information, which include:

1) Color: Presenting different groups in different colors clearly creates some degree of grouping among the elements of the same color. GUIs that utilise color well increase productivity. If like color items are in close proximity, the visual association is stronger than if the like color items are further apart. In addition to changing the item's colors, one can use different colors for the background and foreground. The effectiveness of this technique decreases as the number of screen colors increases. Overuse of color degrades performance, however.

2) Graphical Boundaries: Drawing boundaries around elements is the most common method of grouping elements in GUI. Although there is no empirical evidence to show that these groupings improve performance, users prefer this type of groupings compared to other methods. Another method of grouping is to group tasks within icons. Icon grouping is easy because many icons can have common attributes. Icons are also small and therefore use less space. Another advantage of icons is that recognition is faster for pictures than for text. This makes it easier for the novice to learn a system. Studies also show that icons have smaller error rates than textual interfaces and the same as for menu interfaces. Conversely though, empirical studies have shown that, counter intuitively, icons do not lead to greater increases in performance.

3) Highlighting: Besides color, there are several other methods of highlighting including reverse video, brightness, underlining, and flashing. The most common use of highlighting is reverse video to indicate an item that is currently selected. GUIs usually use brightness to show which items are not active at a given time. Underlining is effective if it does not interfere with the legibility of characters. Flashing will both get attention and annoy if the user cannot turn off the flashing. Therefore, one should use flashing only to convey an urgent need. Apple Macintosh uses flashing to signal only program or data destruction. Regardless of which type of highlighting, one needs to apply it conservatively. Overuse of highlighting causes confusion among users and defeats its purpose. Additionally, if one highlights the wrong information, the user has more difficulty detecting the important information.

Information Sequencing

One needs to lay out a screen in a manner that allows the user to easily find any information on it. Most designers advocate the use of the de facto GUI screen

standards. This is because many users now expect certain modes of operation in all GUIs. For example, most users expect the top of screen to contain the headings for the pull-down menus. The top right is the default location for icons representing the disk availability. In the Macintosh GUI, the bottom right contains the trash icons used for deleting files.

Within a window, there are also many standard modes. A window title is usually at the top. Scroll bars are on the right and bottom for vertical and horizontal window movement. A box for closing the window is at the top left. Icons for resizing the window are at the four corners.

Studies show that most users initially scan the screen starting at the upper-left corner. This corner should be the obvious starting point for applications invoked from within the window. This permits a left-to-right and top-to-bottom reading, which is standard for Western cultures.

The optimum sequence for screen presentations is a collection of various factors, including:

1) Sequence of use: One needs to present the user the information in the order that the user will probably utilise it.

2) Conventional Usage: If a common convention is in general usage, the GUI design should continue using it. For example, in the standard window layout, the file option is usually to the far left of the menu bar.

3) Importance: The designer needs to place the more important information at a prominent location. For example, if several entries are possible, the GUI should lead off with the required ones and end with the optional ones.

4) Frequency of use: One should pace the most frequently utilised commands at the beginning. For example, in a menu list, the most frequency utilised commands should be at the top of the list.

5) Generality versus Specificity: The more general items should precede the more specific items, especially when there is a hierarchical relationship among the data.

6) Alphabetical or Chronological: If there is no other rules for ordering data element, then one should adopt some other technique such as an alphabetical or a temporal listing. Card [11] showed that selection time was faster for alphabetical than for any other functional grouping.

1.8 GUI EXAMPLES

The goal of any GUI is to allow the user to work through the computer and application to concentrate on the primary cognitive task. The user should not be concerned with the user interface. Any attention devoted to the interface interferes with the main task.

This section presents Graphical User Interfaces. Some popular GUIs will be covered to provide a broad picture of the subject. There are a great many popular GUIs around including X Windows, Microsoft Windows, NeXT's NeXTStep and others.

1.8.1 Microsoft Windows (MS-WINDOWS)

MS-Windows is the most popular GUI for IBM personal computers. IBM and Microsoft announced OS/2 as a new operating system for 80286 and 80386 personal computers. The OS/2 Standard Edition 1.1 adds Presentation Manager (PM) for its graphical user interface. The user interfaces of Windows and PM are very similar but their APIs are different. Microsoft's strategy is to use Windows as a springboard for PM.

Windows provides an environment that enhances DOS in many ways. The major benefits of Windows are:

1) **Common Look and Feel:** All windows applications have the same basic look and feel. Once you know one or two Windows applications, it is easy to learn another one.

2) **Device Independence:** Windows presents a device-independent interface to applications. Unlike most of today's DOS applications, a Windows application is not bound to the underlying hardware such as mouse, keyboard or display. Windows shields the application from this responsibility. The application deals with the Windows API to manipulate any underlying devices.

3) **Multitasking:** Windows provides non-preemptive multitasking support. Users can have several applications in progress at the same time. Each application can be active in a separate window.

4) **Memory Management:** Windows also provides memory management to break the 640K limitation of MS-DOS. An application has the ability to use the extended memory, share data segments with other applications and unwanted segments to disk.

5) **Support for existing DOS applications:** Windows allows most standard DOS applications to run under it directly. Any application that does not control the PC's hardware, use the PC BIOS or MS-DOS software interrupts, can run in its own window.

6) **Data Sharing:** Windows allows a data transfer between application Clipboard. Any type of data can be transferred from one window with the clipboard. The Dynamic Data Exchange (DDE) protocol defines how two applications can share information. Information such as bitmap, metafile, character strings and other data formats can be shared.

Support for Object Orientation

In order to create screen objects such as windows, the application developer defines a class (similar to record) specifying the necessary properties. Instances of class can then be created. Several applications can share the same windows simultaneously. To communicate with Instances of a window class, messages are sent and received by a special function called the window function. The windows handle all messages such as re-drawing the screen, displaying icon or pop-up menus and changing the contents of the client area.

Creation and Manipulation of a Window

MS Windows presents a pre-defined style for user-defined windows; it presents the structure of a window, followed by a discussion of windows manipulation.

1) **Structure of a window:** *Figure 7* displays possible elements for a window. The caption bar (or title bar) displays the name of application within the window. The system menu box contains names of commands available in all applications, such as minimise, maximize, resize and close. The minimize box clicked once reduces the window to an icon. The maximize box enlarges the window to the full screen. The menu bar contains a list of commands In the application. The client area is the area inside the window, which is under the application control.

2) **Creating windows:** Each window created on the screen is a member of some user defined window class. Window classes are created by the application program. Several window classes can be active simultaneously. Each window class in turn can have several instances active at the same time. There are no predefined generic window classes that come with MS-Windows.

To create a window the following steps must be taken:

a) **Set up a window class structure** which defines the attributes of the window class. Attributes that can be defined include:

i) the window function, which handles all messages for this class

ii) the icon and the cursor used for this class of windows

iii) the background color of the client area

iv) the window class menu

v) the re-drawing function used when resizing horizontally or vertically.

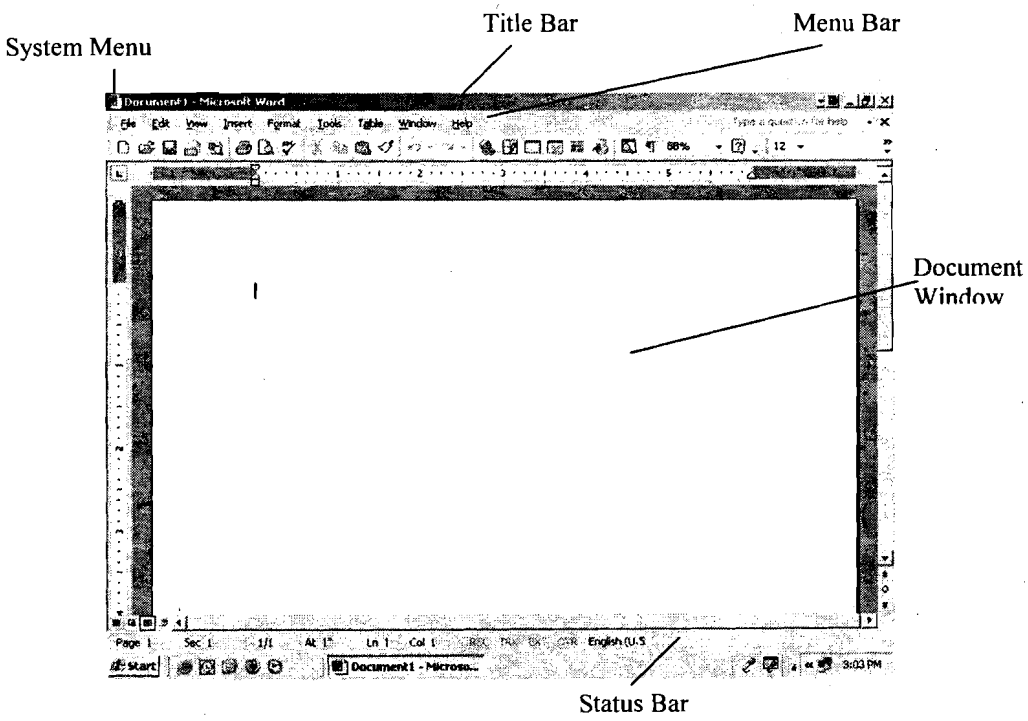

Figure 7: MS-WINDOWS screen element

3) Manipulating windows: An application can choose to display the Windows, resize the window, display additional information in the client area, and so on.

Pop-up and Child Windows

Pop-Up and child windows are special type of windows, and are used to communicate information between the application and the end-user. They remain on the screen for a short period of time. In this example, the pop-up Display information about a given file such as date and time of creation and the size. Dialog boxes, as discussed earlier, are a more sophisticated form of pop-up windows.

MS-Windows provides a collection of predefined child windows. These are the most common usage of child windows. These predefined classes are button, scroll bars, listbox, edit and static class. A developer can also define child windows that can be controlled by any user-defined operation.

Resources

Resources are used to manage windows and user-defined object. **MS-WINDOWS** provides nine kinds of resources to application developers. These resources are : icons, cursors, menus, dialog boxes, fonts, bitmaps, char strings, user-defined resources, and keyboard accelerators.

1) **Icons and cursors:** Windows defines a few types of icons and cursors. An icon or a cursor is essentially a bit-mapped region that is used to represent and symbolise a window or cursor. A developer can also define an original icon or cursor using the ICONEDIT utility.

2) **Menus:** Each window can have its own menu bar. A menu item can be a character string or a bitmap. Each item of a menu bar in turn can have a pop-up menu presenting a list of options. Currently, Windows does not support nesting of pop-up menus within other pop-up menus. (Windows 3.0 provides this functionality). But a pop-up menu can invoke a dialog box. When a menu is selected, Windows sends one or more messages to the Window function of the Window containing the menu bar. These messages can be interpreted to perform the function corresponding to the menu item.

3) **Dialog boxes:** These provide another mechanism besides pop-up menu and menu bar to obtain information from the end-user. Dialog boxes are much more flexible than menu bars or pop-up menus. Dialog boxes usually contain a group of child windows such as buttons, scroll bars, and editable fields. Just like windows, dialog boxes have a function that is used to process messages received from the user upon selection of options. Generally, dialog boxes appear as pop-up windows. The user selects the option needed from the dialog box and the dialog box disappears. *Figure 8* depicts an example of a dialog contains an edit box, a list box, and open and cancel buttons. The end-user can specify the name of a file either by selecting from the list box or by typing the name of the file in the edit box. By clicking on the open button, the application will open the selected file.

4) **Fonts:** Windows provides a few families of fonts with different sizes and shapes: Modern, Roman, Swiss, Helvetica, and Script. Application processors and desktop publishing can define additional fonts as needed.

5) **Bitmaps:** They are used to represent icons, cursors, or draw picture on the screen. Both mono and color bitmaps can be defined.

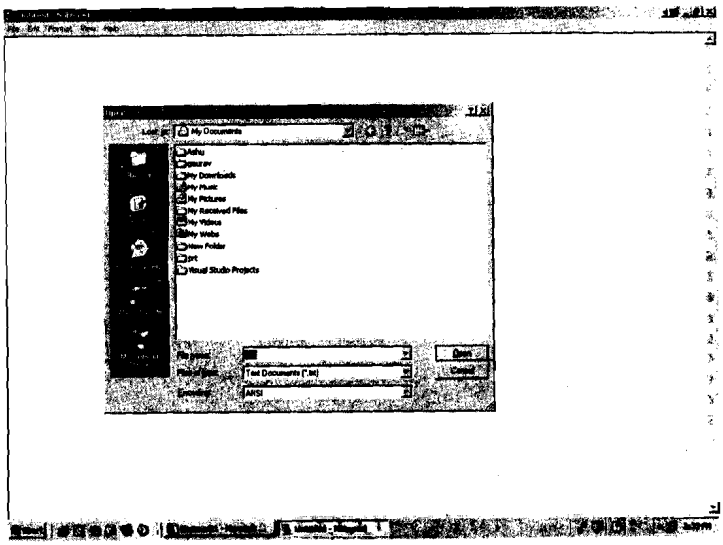

Figure 8: Dialog Box (MS-Windows) with an edit box, a list box, and push buttons

6) **Character strings:** Character strings are handled as resources mainly to provide a manageable solution to internationalisation of a window application. Instead of including a character string as part of the code, it can be placed in a resource file handled as a resource. Once in a resource file, different versions of

the same character string resource file can exist for different languages. This separation of the character strings from the code makes internationalising the application much easier, and also makes maintaining the application much simpler.

7) **User-Defined Resources:** These can be used for many purposes and support any user-defined data type. Any arbitrary data can be managed as a user-defined resource.

Resources are defined in a text file called a resource script. They are compiled through a utility called the Resource Compiler and linked with the windows application. Resources are read only data. Once a resource script is compiled, it can be shared with other window applications.

Graphics Device Interface

Graphics in Windows are handled by the Graphics Device Interface (GDI). Both vector and raster color graphics are supported by GDI. GDI supports only two-dimensional graphics. Vector graphics are supported by a set of vector drawing functions such as drawing a line, point, and polygon etc. pie chart.

Raster graphics are supported by pixel manipulation. Raster graphics can be stored or manipulated either as a bitmap or as a metafile. A bit-mapped representation of the graph can be manipulated using utilities such as BitBlt, PatBlt, and StretchBlt. A metafile provides binary encoding of GDI functions such as to draw vectors and to fill a region with a bitmap. Metafiles take up less disk space than a bit-mapped representation since they do not represent each pixel directly on disk. When metafiles are played, they execute the function encoding and perform the necessary graphics operations to display the graphics output.

☞ **Check Your Progress 2**

1) What are the four major components of GUI? Explain the functioning of any one component.

.....

.....

.....

2) How does MS-Windows enhance DOS environment?

.....

.....

.....

3) How are graphics supported in MS-Windows?

.....

.....

.....

1.8.2 Macintosh Toolbox

The tremendous success of the Macintosh computer popularized the window-style menu-driven user interface. Macintosh got its early start from Apple's Lisa.

The Macintosh GUI is called the Toolbox. The Toolbox consists of a collection of utilities to manipulate Macintosh's resources. In this section we present an overview of Toolbox functionally, examine the object-oriented features of the Toolbox and discuss major components of the toolbox's user interface features.

Functional Overview

The Toolbox provides a collection of utilities to access and manipulate Macintosh hardware and software resources. It provides a set of utilities to manipulate interface components such as windows, menu bar, and dialog boxes. These are discussed in the following sections. Some of the other utilities provided are:

1) **Fonts Manager:** allows manipulation of system and user-defined fonts.

2) **Event Manager:** provides monitoring of events generated by keyboard and keypad.

3) **Desk Manager:** provides access to desk utilities such as the calculation.

4) **Text Edit:** provides simple text editing capabilities such as cut and paste.

5) **Memory Manager:** provides a set of routines to manipulate dynamic memory.

6) **File Manager:** provides a set of routines to manipulate files and transfer data between files and applications.

7) **Silver Driver:** provides access to sound and music capabilities.

8) **Toolbox Utilities:** provides a set of general routines to manipulate icons, patterns, strings, fixed-point arithmetic, and so forth.

The Toolbox can be manipulated either from the assembly language or Macintosh Programmer's Workshop (MPW) language (Pascal, C, C++). Toolbox can be accessed from non-MPW languages including Think C. One of the major features of the Toolbox is that most of the Toolbox utilities reside in ROM. This provides a very responsive user interface.

From this point on, we will concentrate on the user components of the Macintosh Toolbox such as the Window Manager, the Manager, the Menu Manager, the Dialog Manager, the Scrap Manager, the Draw, and the Resource Manager.

Object-Oriented Features of Toolbox

Just like Microsoft Windows, the Macintosh Toolbox design is object-oriented. For example, the application developer defines a new type of window by the template for a new class of windows. Messages are sent between Toolbox application to change the behaviour of screen objects such as windows and cursors. When a window needs to be drawn or resized, a message is sent to the **window definition function** to perform the appropriate action.

The Window Manager

The Window Manager allows the application developer to manipulate windows on the screen. It deals with issues such as overlapping windows, resizing windows, moving windows around the screen, or changing a window from a foreground to a background window. Developers can create and manipulate pre-defined Toolbox windows with any desired shape and form. For example, the user can create a circular or a hexagonal window.

Toolbox comes with a series of pre-defined windows. The most popular of these windows is the document window. The document window, depicted in *Figure 9,* has

the following regions: Title bar, Close Box, Scroll bar(s), Size Box and content region. The Title bar contains the title of the window. By clicking and holding the mouse within the region, the window can be dragged to a different screen. The Close Box is used to shut the window. Clicking inside this box will cause the window to disappear. The Size Box is used to resize the window. The horizontal or vertical Scroll bars can be used to scroll the contents of the window. The content region is the area that is controlled by the application.

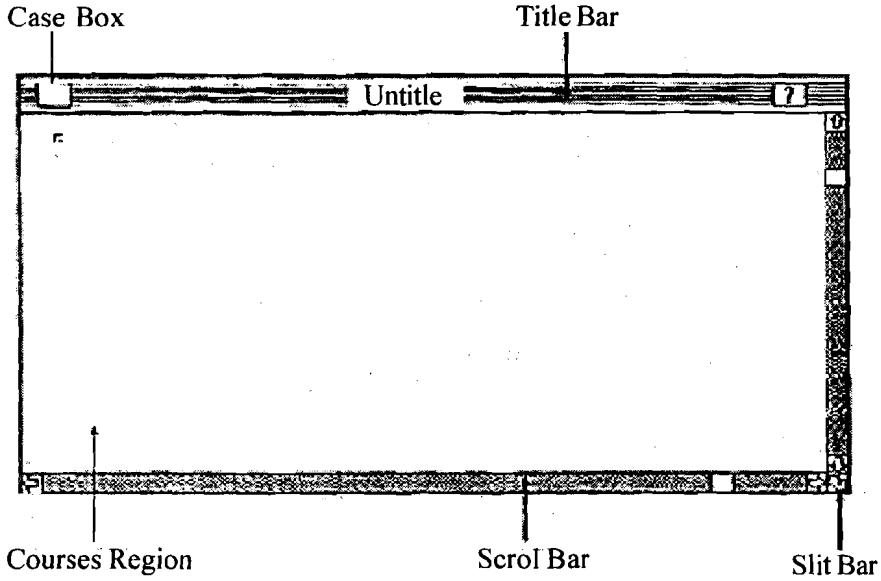

Figure 9: Macintosh Toolbox Document Window

When creating a document window anyone of the components described above can be omitted. For example, anyone of the Scroll bars can be absent from the document window.

New types of windows can be created through either the Window Manager or the Resource Manager. The Resource Editor can be used to create a window template. A window template essentially defines a class of windows that can be instantiated by an application. For each class of window, defined by the system or there exists a function called the **window definition function**. This function is used to handle the behaviour and appearance of the window. When a window needs to be drawn or resized, a message is sent to the **window definition function** to perform the action for that type of window.

The Window Manager provides a large collection of routines to create, display, hide and activate/deactivate windows on the screen.

The Resource Manager

The Resource Manager provides access to various resources such as the windows, fonts, icons and menus. A user-defined resource such as a window is defined as Resource Editor. The template of the window and the window definition function is stored in a file called the resource file. A unique resource identifier is used to access a pre-defined resource. Resource identifier can be used to recognise the type of resource and the actual resource file to be read.

The Menu Manager

The Menu Manager provides a set of routines to create and manipulate menus. A menu bar, depic ted in *Figure 10,* can contain a list of one or more menu items. Each menu item highlights itself when clicked with the mouse; this item is selected. The developer can define a menu using the standard bar. Menus can be stored as a resource and managed by the Resource Manager. Once given to Manager, a unique identifier is used to reference the menu.

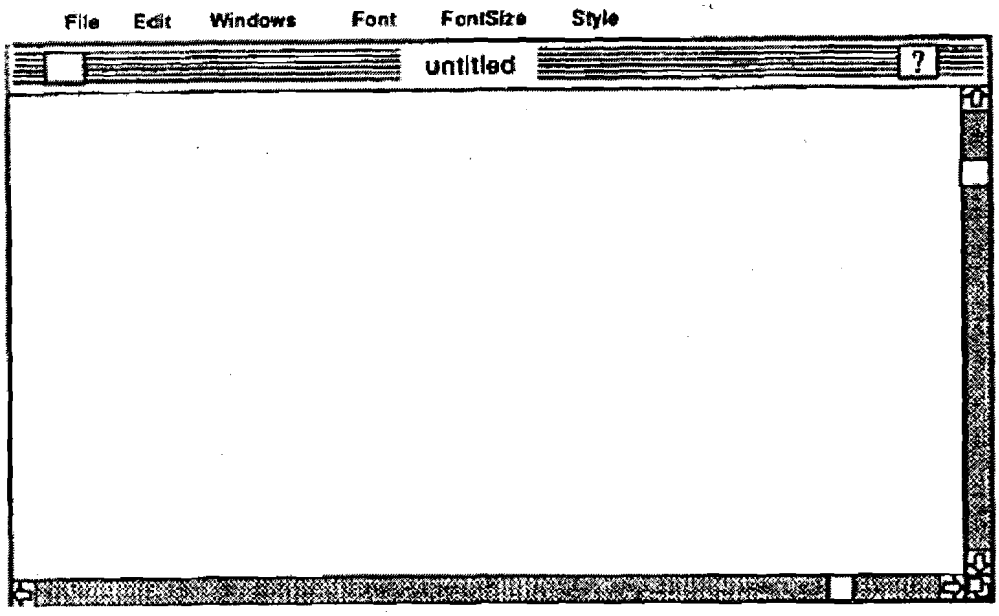

Figure 10: Macintosh Menu Bar

A menu item can even invoke a pull-down menu, which can be used to or choose additional selections, see *Figure 11*. A menu item in a pull-down menu can then invoke a dialog or an alert box. A menu item can be a textual icon. Unlike MS Windows, menus in Toolbox cannot be a part of a window Menus and only used to define the menu bar, which is associated with one Macintosh screen.

The Control Manager

The Control Manager provides a set of routines to define objects like buttons, check boxes and scroll bars. Controls are usually a window. For example, the scroll bars can be defined as part of a document. Most often controls are defined as part of a dialog or an alert box.

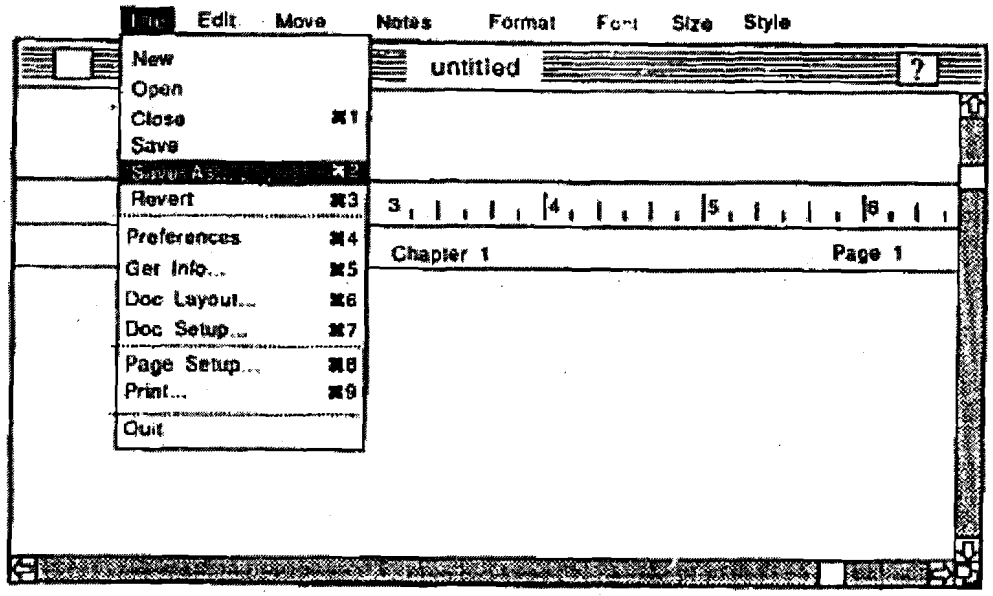

Figure 11: Macintosh Pull-down Menu

The Dialog Manager

The Dialog Manager provides a set of routines and data structures to manipulate dialogs and alert boxes. Dialog boxes are used to get additional information from the end-user of an application. An alert box is used to provide information such as a warning or a cautionary message.

The Scrap Manager

The Scrap Manager allows the user to move data from one application to another. A portion of one window can be cut and placed in the scrap manager and then pasted into another window. Once a portion of a window is cut, it is placed in the clipboard. The user can paste the contents of the clipboard into another window.

1.8.3 X-Windows

The name X, as well as part of the initial design, was derived from an earlier window system called W developed at Standford University. Currently, the X Window system is supported by an X Consortium of primarily UNIX-based hardware and software vendors as a standard base for user interfaces across most UNIX product lines. *Figure 12* illustrates the components of UNIX GUIs based on X Windows.

The X Window system does not define any particular style of interface but rather provides a mechanism for supporting many styles. X is also network-based. In contrast with PM, the X architecture is based on the premise that an application can run one computer, while the graphical presentation of the application's output and responses from the user can occur on another computer. As such, X provides a flexible set of primitive window operations but carefully avoids dictating the look and feel of any particular application. X's device-independent functionally allows it to serve as a base for a variety of interface styles. X does not provide user interface components such as buttons, menus, or dialog boxes.

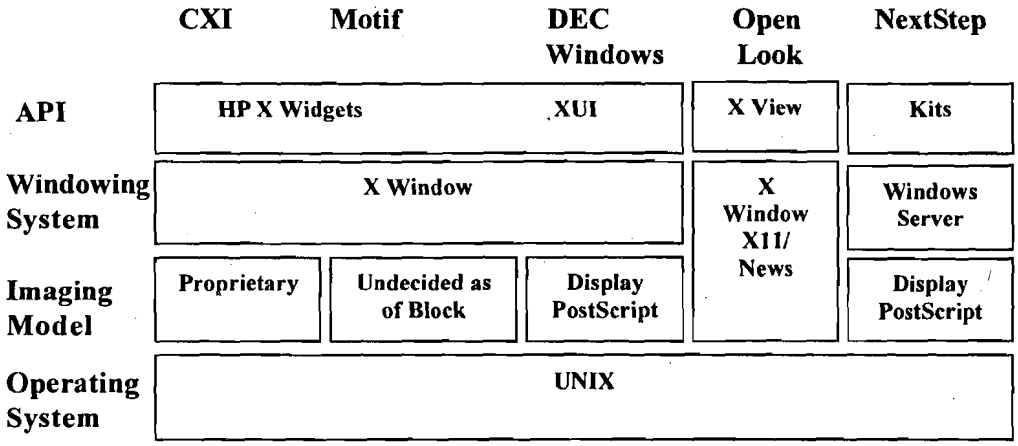

Figure 12: Components of major UNIX-based GUIs

An interface supporting X can theoretically use any X-Window display. The application program sends calls to the X-Window Library, which packages the requests as X packets and sends calls to the X-Window server. The server decodes the X packets and displays them on the screen. Exactly how the packets will be displayed (i.e. will look) on a workstation depends on the set of pre-designed window elements called widgets that are unique to the particular workstation in the system environment. The ultimate look on one workstation may differ substantially from that on another, but the response to a user's action on these different-looking interfaces will be the same. Several hybrids of X exist.

The application-programming model for X, like PM and Windows, is event-driven, and the categories of events are similar. Like PM, X programs rest in wait loops until the underlying window-management system generates an event. Window hierarchy management is also common to the architecture of both systems. Like PM's window, windows in X are organized as parents and children, so attributes can be inherited and effects of the events applied to related members.

Since X does not support any particular interface style, most programmes build applications with libraries that provide an interface to the base window system. One standard programming interface to X is the C language library known as X Library or Xlib. This library defines functions for access and control over the display, windows and input devices. Although commonly used, the library can be difficult and tedious to work with and often results in hundreds of lines of code for a GUI. A better solution is a higher-level toolkit, such as the X Toolkit (Xt), upon which many other X Window toolkits are based.

X Toolkit is the foundation upon which most of the commercial toolkits are based and is a good example for describing object-oriented functionally. The X Toolkit consists of two parts: a layer known as Xt Intrinsics and a set of widgets. Xt Intrinsics supports many of the available widget sets. The widget set implements user interface components including scroll bars, menus and buttons. Xt intrinsics provides a framework that allows the programmer to combine the components. Both parts are integrated with Xlib, allowing programmers access to higher-level libraries. A typical X programming environment is illustrated in *Figure 13.*

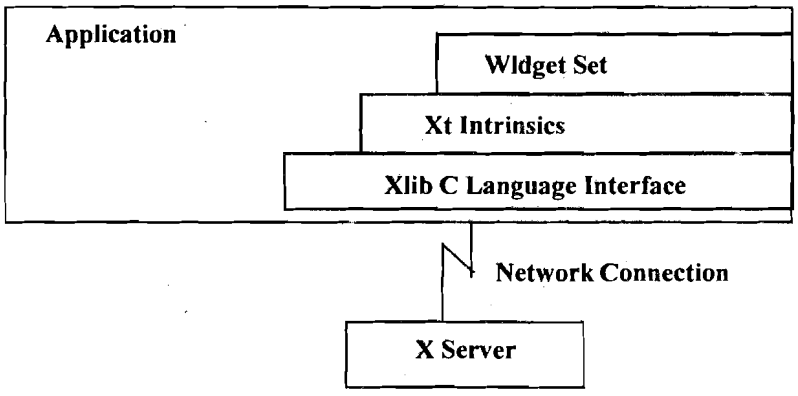

Figure 13: Typical X Window Development Environment

The common distinction in object-oriented programming between an application programmer (consumer) and a class library programmer (producer) fits well into the X environment. The X application programmer is likely to use a combination of a higher-level toolkit (e.g., the Toolkit's Xt Intrinsic), Xlib, and a widget set. The class library programmer, whose job is to create new widgets, is likely to use the capabilities from within one of the higher-level toolkits.

Depending on the toolkit, the widget programmer may be able to take advantage of object-oriented capabilities. Xt Intrinsic, for example, uses an object-oriented approach and organises widgets into classes. Xt Intrinsic defines the basic architecture of a widget. This allows widgets to work together smoothly when built by different programmers or potentially when selected from different widget sets.

A typical widget consists of two basic parts: a class record and an instance record. Each of these components is implemented as a C structure containing data and pointers to methods. Intrinsics defines the organisation of each structure. All widgets belonging to a class share a copy of common data methods for that class. Each individual widget has its own copy of instance-specific data. A widget's class record is usually allocated and initialised statically at compile time; a unique copy of the instance record is created at run time for each individual widget.

Since all widgets belonging to the same class share the same class record, the class record must contain only static data that do not relate directly to the state of an individual widget. For example, every widget's class record includes a field containing the widget's class name. The class record also contains methods that define the

appearance and behaviour of all widgets in the class. Although most of these methods operate on the data in the widget's instance records, the methods themselves are shared by all widgets in a class.

Many object-oriented languages provide inheritance as a language construct. The Xt Intrinsics is written in the C language, which does not directly support object-oriented programming. Xt itself supports inheritance with a subclassing mechanism. To use Xt's subclassing capabilities, the programmer first finds an existing widget with similar functions (a common step in object-oriented design), writes a subclass that can inherit existing data and methods, and then adds new data and methods as needed.

If a similar widget cannot be found, then a foundation class called Core is subclassed. All widget classes are subclasses of the core widget class. Like the class Window in PM, Core contains the basic methods for initializing, displaying, and destroying widgets, and reacting the external resizing. Core also stores basic properties (e.g., the geometry) of widgets and the data for handling events.

New widget classes inherit the methods (called resources) defined by their superclass by specifically including the definition of the superclass structure in the definition of the new class. Xt Intrinsics provides two mechanisms for inheriting the methods defined by a superclass. The first mechanism is referred to as chaining. When a method is chained. Xt Intrinsics invokes the method defined by a widget's superclass first and then invokes the widget's method. This allows a widget to inherit part of a method from its superclass.

Xt Intrinsics also provides a mechanism for inheriting methods that are not chained. This is done by using special symbols to specify methods in the widget's class records. Each symbol is defined by the superclass that added the method to the widget's class record. These symbols can be used by a subclass of the widget class Core and do not have to be redefined by each widget class. Only classes that contribute new methods to the class record need to define new symbols. When a widget class specifies one of these symbols in the class record, Intrinsics copies the corresponding method used by the widget's superclass into the widget's class structure at class initialisation time. *Figure 14* illustrates this architecture and shows the relationship between the class record and instance records of several widgets belonging to widget class core.

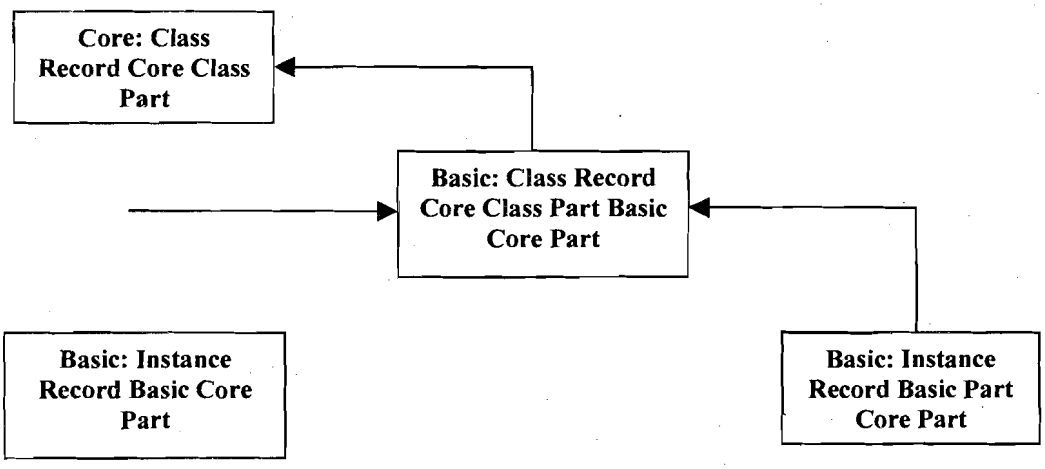

Figure 14: Inheriting from Widget Class Core

The Xt Intrinsics uses a data-abstraction technique to hide the implementation of a widget from applications that use the widget. Applications that use a widget see only the incomplete definition of the widget and therefore cannot directly access fields in the widget structure.

Applications declare all widgets as type Widget, which is known as opaque type. This means the application has a pointer to the widget structure but does not have access to the real definition of the data that it represents. The application cannot access the contents of the data structure.

Another object-oriented technique promoted in X Windows is to extract some general functionality from two or more new widget classes and create a metaclass. A metaclass is an abstract class that is not intended to be instantiated directly but serves as a superclass for other similar widgets. When creating a complete new widget set, it is frequently useful to create a class of hierarchy or metaclasses. With this approach, the top metaclass in the hierarchy defines elements that all widget classes in the set have in common, and each subclass becomes more and more specialised. Such an organisation allows the widget programmer to create new widget classes with the least amount of effort, although there is some extra initial effort required to design and create metaclasses. For example, this approach is used by the X Widget set from HP, where most basic widgets inherit from the primitive metaclass and most composite widgets inherit from the Manager widget class.

☞ Check Your Progress 3

1) What types of utilities are provided in Toolbox? Explain their features.

.....

.....

.....

.....

2) Explain the functioning of Resources Manager and Menu Manager of Toolbox.

.....

.....

.....

.....

3) What is the basic philosophy of X-windows? How is it different from the rest of GUIs?

.....

.....

.....

.....

1.8.4 NeXT

The NeXT computer, with its three-dimensional user interface, was introduced in 1988. It has grabbed the attention of the computer industry. The machine has been hailed as the most innovative computer invented in recent times. The computer was initially intended for the educational market. But the NeXT Corporation decided to widen the market for its machine to the commercial arena.

We provide a brief overview of the NeXT software tools and capabilities, followed by a discussion of the capabilities of the user-interface design.

Overview of NeXT Software

The NeXT computer is designed to reach a wide range of users from non-technical to power users. The non-technical users can deal with the graphic user interface to perform

tasks by manipulating menus, icons, and dialog boxes. The power user can directly interact with its Mach Operating system.

The NeXT system software comprises three major pieces: the Mach operating system, applications and the NeXT user interface. The Mach operating system, developed at Carnegie Mellon University, is a re-designed UNIX. Mach re-designs the UNIX kernel to reduce the size.

NeXT also comes with a set of bundled applications. Currently, there are new applications being developed that take advantage of NeXT hardware and software capabilities. Some of the applications supported on NeXT are NeXT SQL Database Server, Mathematica (symbolic mathematics package, WYSIWYG editors and word processors, and more.

NeXT User Interface

The third and last component, the NeXT user interface, is the most impressive piece of NeXT technology. The NeXT user interface draws heavily on direct manipulation and modern graphic user interfaces. The NeXT user interface is composed of four components. Workspace Manager, Interface Builder, Application Kit, and NeXT Window Server.

The Workspace Manager allows the user to manage files and directories and to execute programs. When a user logs into a NeXT machine, the Workspace Manager is started, *Figure 15.* The Directory Browser window is used to navigate through the files on disk.

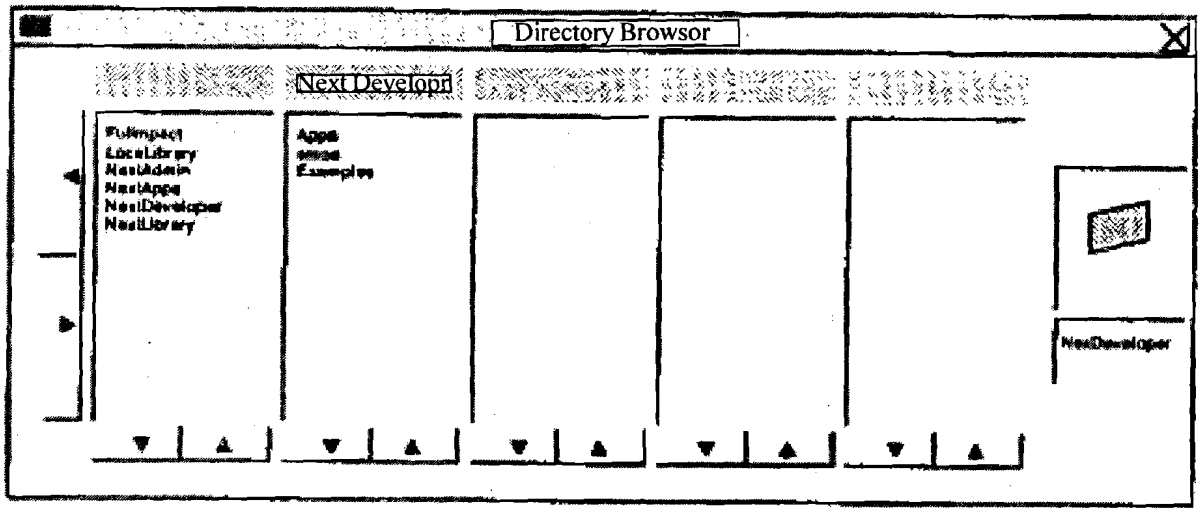

Figure 15: NeXT Workspace Manager Directory Browser

The Interface Builder lets the user create interfaces on the screen without writing a single line of code. Users simply select the menu, control and screen objects from a palette, and then move the controls to the desired location.

The Application Kit is a library of user-interface objects. It is used in conjunction with the Interface Builder to design user interfaces. The Application Kit is discussed in the next section.

The Window Server handles all screen activities, such as drawing windows and handling events such as mouse clicks. The window Server itself does not perform the drawing and screen I/O commands. Display PostScript, designed by Adobe and NeXT, handles all such activities. Up to now, PostScript has been used only as a language for printer engines. With the advent of the PostScript, both screen and printer share the same protocol. Therefore one drawing method is used to display objects on the screen and the printer.

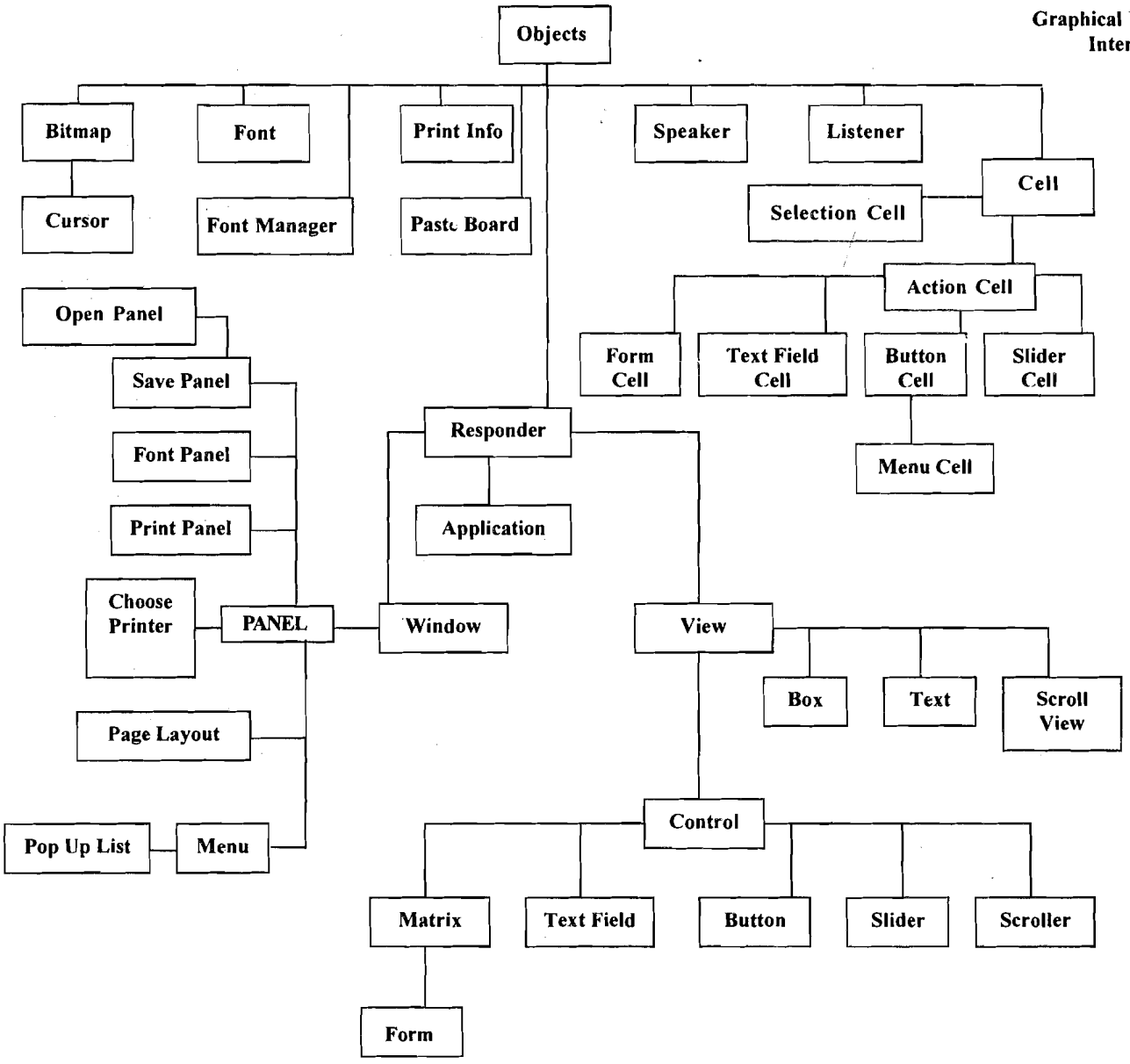

Figure 16: Application Kit

Application Kit

The Application Kit provides an extensive library of predefined classes. The hierarchy of classes is shown in *Figure 16.* These classes provide functionality to define user interfaces composed of menus, windows, buttons, slide bars, and sound. Each class within the hierarchy defines the behaviour of its objects. For example, a menu knows how to handle mouse events, when clicking on a menu item. Window objects know how to resize the window.

The Application Kit can be extended by adding new subclasses to the hierarchy. To add new subclasses, the class definition is written using the Objective C language. Each NeXT machine comes with a copy of Objective C. Objective C is an object-oriented extension of C. The language is a superset of C incorporating object orientation features from **Smalltalk**. Just like Smalltalk, it comes with a large collection of predefined classes to simplify the software development task. The language supports abstract data types, inheritance, and operator overloading. Unlike C++, Objective C does not extend the definition of any existing C language construct. It relies totally on the introduction of new constructs and operates to perform tasks such as class definition or message passing.

To develop user interfaces, designers can use Application Kit from Objective C directly. This would be very much like developing an application using MacApp. They can create instances of objects from the Application Kit hierarchy and modify the

attributes by calling the methods attached to the class definition. But the other method of defining user interface, using the Interface Builder, is much easier than coding it entirely in Objective C.

Designing User Interfaces with Interface Builder

The Interface Builder provides an easy to use utility to design a user interface using the muse and the screen. The Interface Builder is similar to an icon editor or screen painter. Designers can define the screen layout by selecting the screen objects from the Interface Builder palettes. The Interface Builder also helps to define the user-defined class and make connections between objects. Not all of the coding is automatic; the Application Kit and the Objective C language are also needed.

Defining a user interface using the Interface Builder requires the following steps:

1) **Define layout of screen:** the interface designer defines a user interface by simply selecting screen objects from the Interface Builder palettes, see *Figure 17.* After picking an object from a palette using the mouse, the object can be dragged into the destination window and resized as desired.

   The Interface Builder palettes include objects such as windows, menus, buttons, fields radio buttons and more. At the top of the palettes window, three buttons allow the interface developer to select a wide array of user-interface objects.

2) **Define the user-defined classes:** the developer defines a new class definition using the Classes Window. The Classes Window allows the developer to extend the Application Kit class hierarchy. The developer navigates through the class hierarchy and creates a new subclass within the hierarchy. Methods and outlets (see next step) are defined for this new class definition. When a class is defined this way, only the template of the class is created.

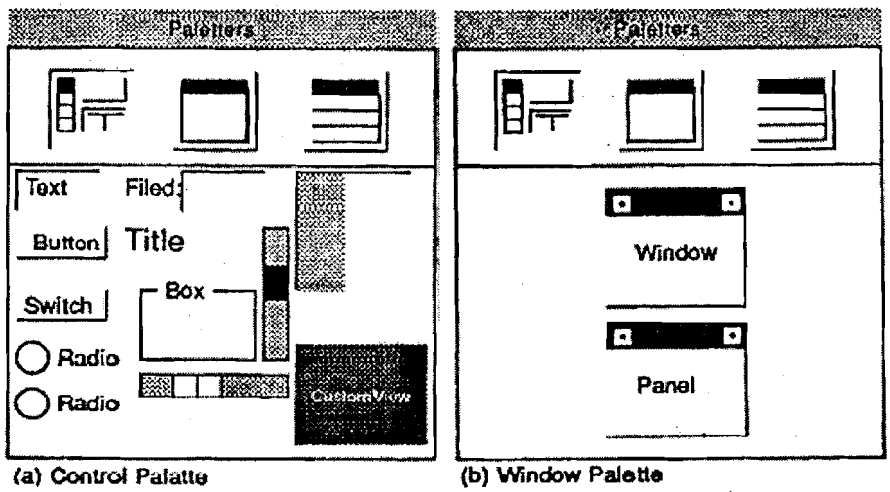

Figure 17: NeXT interface Builder palette

3) **Making connections:** up to this point we have defined the layout of the user interface. At this step, the developer needs to make connections among application objects. For example, when a scroller's slide bar is moved, a message is sent to an application object to perform certain actions like shifting the text. Again the Inspector is used to connect user interface objects with methods of classes within the application.

4) **Actual application code:** the previous steps are handled by Interface Builder directly. The last step is accomplished by writing the application code in Objective C. When the developer is done with the first two steps, the Interface Builder defines the source files necessary to build the application. These files contain the template for the class definitions and the connections made among objects. At this stage, the developer needs to extend these source files. Extensions are made to specify the logic of the program and the code for method definitions.

☞ **Check Your Progress 4**

1) What are the major components of NeXTSTEP? How do these elements

.....

.....

.....

2) How are applications written in NeXTSTEP environment?

.....

.....

.....

1.9 SUMMARY

The GUI has already made a tremendous contribution to the increased usability of computer systems. It is with great excitement that we look forward to future innovations in human-computer interfaces. GUI development is at the vanguard of creativity in many areas, such as ergonomics, software development tools, computer graphics and linguistics to name just a few. The past decade has seen a rapid change in the understanding and definition of GUIs, but we have only been through the infancy of GUIs. There remains much to be done in terms of increasing the productivity of computer users, standardising operations across different architectures and adapting the human-computer interface to non-traditional applications. In this unit we discussed several issues related to GUI including functioning of several popular packages.

1.10 SOLUTIONS/ANSWERS

☞ **Check Your Progress 1**

1) GUI is a system that allows creation and manipulation of user interfaces employing windows, menus, icons, dialog boxes, etc.

   The basic features of GUI are:

   - Pointing device such as mouse, which controls cursor.
   - Point and shoot functionality under control of device, which cause screen menus to appear or disappear.
   - Support of windows, which graphically display the status of a computer program.
   - Icons that represent files, directories and other application and system entities.
   - Support of graphical metaphors such as pull-down menus, dialog boxes, buttons, slides that let the programmer and user tell the computer what to do and how to do it.

2) a) When a screen is split into several independent regions each one is called a window. Several applications can display results simultaneously in different windows. The user can switch from one window to another window. Windowing systems have capabilities to display windows either tiled or lapped. Users can organize the screen by resizing the window or moving related windows closer.

b) No model answer is given.

c) No model answer is given.

d) No model answer is given.

3) Bit-mapped display is made up of tiny dots (pixel) that is independently addressable and has much finer resolutions than character displays. Bit-mapped displays have advantages over character displays. One of the major advantages is of graphic capability.

Check Your Progress 2

1) There are four major components of GUI.

i) A windowing system

ii) An emerging model

iii) An application program interface

iv) A set of tools and frameworks for creating interfaces and developing integrated application.

The API (Application Program Interface) is a set of programming language functions that allow the programmer to specify how the actual application will control the menus scroll bars and icons that appear on the screen. Like in windowing models, APIs align with particular GUIs. The features vary from package to package.

2) There are several ways MS-Windows enhance DOS environment.

i) Device independence. It presents a device independent interface to applications. Unlike most of today's DOS applications, a window application is not bound to the underlying hardware such as mouse, keyboard or display windows.

ii) Multitasking. Users can have several applications in progress at the same time. Each application can be active in a separate window.

iii) Memory management. Windows also provide memory management, the 640K limitation of MS-DOS. An application has the ability to extend memory and share data segments with other applications.

iv) Data sharing. Clipboards allow data transfer between application clipboard. Any type of data can be transferred from one window to another through the clipboard.

3) No model answer is given.

Check Your Progress 3

1) The toolbox provides a collection of utilities to access and manipulate Macintosh hardware and software resources. It provides a set of utilities to manipulate windows menu bar and dialog boxes. Some of the other utilities provided are:

i) Fonts manager: allows manipulation of system and user defined fonts.

ii) Event manager: provides monitoring of events generated by keyboard and keypad.

iii) Text edit: provides simple text editing capabilities such as cut and paste.

iv) Toolbox utilities: provides a set of general routines to manipulate strings, fixed point arithmetic and so forth.

2) No model answer is provided.

3) The X window system does not define any particular style of interface but provides a mechanism for supporting many styles. X is also network-based in contrast with other GUIs, or the X architecture is based on the premise that an application can run on one computer while the graphical presentation of the application's output and the responses from the user can occur on another computer.

Check Your Progress 4

1) No model answer is given.

2) No model answer is given.

1.11 FURTHER READINGS

1) *Communication of ACM*, April 1993.

2) *Object Orientation: Concepts, Languages, Databases. User Interfaces*, Khosafian, Setrag & Razmik Abnours, New York; Wiley & Sons, 1990.

UNIT 2 INTRODUCTION TO OPERATING SYSTEM

Structure **Page Nos.**

2.0 INTRODUCTION

An operating system is a system software which may be viewed as an organized collection of software consisting of procedures for operating a computer and providing an environment for execution of programs. It acts as an interface between users and the hardware of a computer system.

There are many important reasons for studying operating systems. Some of them are:

1) User interacts with the computer through the operating system in order to accomplish his task since it is his primary interface with a computer.

2) It helps users to understand the inner functions of a computer very closely.

3) Many concepts and techniques found in the operating system have general applicability in other applications.

The introductory concepts and principles of an operating system will be the main issues for discussion in this unit. The unit starts with the basic definition and then goes on to explain the stages of evolution of operating systems. It further gives details of several approaches to operating system design.

In the last two subsections of the unit we classify an advanced operating system and explain some characteristics of modern operating systems.