STRUCTURED METHODOLOGIES WITH THE X WINDOW SYSTEM X11/RELEASE 4 AND ORACLE RDBMS 6.0

Stanley J. Sewall,* Internal Revenue Service

Abstract

Windowing systems have become a standard software component with the purchase of most operating systems. The X Window System from Massachusetts Institute of Technology (MIT), has gained recognition in the computer community as a viable desktop graphical user interface (GUI) environment. This paper will discuss the implementations of ORACLE RDBMS 6.0 and Oracle Tools (SQL*Forms and Pro*C) with the X Window System X11/Release 4. Knowledge in both areas is a prerequisite to ensuring an efficient development platform.

Introduction

The X Window System bestows the Oracle programmer with a rich control over his/her work environment. The ORACLE RDBMS 6.0. provides the programmer with an expeditious, secure, and centralized storage medium. In addition, Oracle's tools, such as SQL*Forms, SQL*Plus, and Pro*C, are separated from the centralized database. This separation allows for flexibility when engineering applications.

The unification of the two work environments resulted in a number of benefits. These include:

- the normalization of functions within applications
- the standardization of applications between projects
- the ability to develop code from a distributed database
- a 'Look-and-Feel' quality on a bit-mapped X terminal
- the ability to develop applications faster, without creating more errors in the development phase of the life cycle

*This paper was originally presented at the 1992 International Oracle User Week Conference in San Francisco, California, September 14-18, 1992.

An encapsulation tool, such as the X Window System, "wraps around" existing character-based applications. The X Window System monitors output and allows the construction and modification of X Window programs to interface with old application displays.

Paper Overview

This paper will focus on current application of the X Window System at the Internal Revenue Service's service center in Cincinnati. First there will be a brief overview of the X architecture. Then some application methodology will be presented. Discussion of implementation will include:

- assembling the programming team;
- adapting to the new environment;
- coping with behavior and processing differences between the new and old systems; and
- fine-tuning the Oracle - X Window System.

The paper will conclude with some possible considerations for users and developers.

X Architecture Overview

The architecture of the X Window System is based on the client-server model. The system is separated into two distinct parts: the display servers, which provide window properties and monitor user input, and the X Window applications, called clients.

The division within the X architecture allows the clients and the server either to be executed on the same host or to reside on different machines (possibly of different types, with different operating systems).

The X display server, usually called the X server, is a program to monitor the input and output devices. As the server receives information from a client, it updates the appropriate window on the display. The X server sends and receives commands in the form of packets. Packets are information of an event that has transpired within a

window and is sent to and from the server. The X server advantage working with the client-server model is: since the server is entirely responsible for interacting with the X terminal, only the server program must be machine-specific. X Window System allows the user to execute several clients simultaneously. For example, the user can edit a text file in one window, consult his or her calendar in a second window, read mail in a third, all while displaying the system load averages in a fourth window.

X client applications communicate across the network with the display server by means of calls to a low-level library of "C" Language subroutine in file Xlib.h, which resides in the /usr/lib/include/X11 directory. Xlib.h library provides functions for connecting to a particular server, creating windows, drawing graphics, and responding to synchronous events, among others.

Application Methodology

The X Window System was introduced at the IRS processing center to provide statistical taxpayer information. The Oracle programming methodology before the acquisition of the X terminals consisted mostly of field-to-field consistency testing, in which data for each field were checked to verify that they met certain tolerances established for that field. Needless to say, this approach was quite cumbersome. Also, the field-to-field methodology performed substandard in benchmark testing due to limited memory capacity resources (Program methodology design will be discussed later.)

Now, with the X Window System, the users have character-based Digital VT420's for editing and viewing SQL*Forms applications and batch reports. The programmers use NCD 14c color X terminals. The X terminals purchase was to migrate toward Oracle CASE*Tools in the near future. One aspect of the unique application design is that SQL*Forms can now do preliminary consistency testing before executing the main program. Currently, the programmers are using SQL*Forms as a menu screen to examine one record from the database. The SQL*Forms menu methodology elevates database querying and updating of multiple records. This methodology allows for data discrepancies to be found more easily by the database designers and programmers.

Another advantage which should be mentioned is that Pro*C applications can be executed without modification in the X environment. The applications are executed in a batch mode for the user. The user must access the SQL*Forms application programmed to receive parameters and transfer the input to the batch application program with an Oracle 'HOST' command.

Implementation

Implementation of a X Window System will be described. This section is divided into five subsections which focus on different subjects of IRS applications of the X Window System. New users may find the information helpful when introducing the system to their own environment.

Programmer Concurrence

When assembling an application programming team, the optimum number of personnel is four people, depending on the size of the project. One person with Oracle proficiency is the Project Leader. The second person should have an intrinsic understanding of the X Window System architecture. The other two programmers should interact and capitalize on the experience of the specialized programmers. In addition, a database administrator should assist the programmers with any system issues encountered during the life cycle of the project.

With full participation among the programmers, it is recommended that the following objectives be implemented:

- Declare any programming considerations, for example the initialization of 'GLOBALs,' before working with the two environments. (These need to be discussed within the group for agreement.)
- Establish realistic project deadlines.
- Develop procedures throughout the team to compensate for the Oracle - X Window System platform, so as to accommodate the individual styles of the programmers.
- Find and augment faulty implementations of past projects in the X Window System.
- Discuss implementation and design process early in the project. (Normalization of application design can be achieved by using prototype to assist programmers with X window behavior.)

When designing a system, an application prototype is advisable to indicate Oracle application behavior in the X Window System. Testing procedures are compulsory to ensure the Oracle - X Window System environment is faultless in data capture and interpretation.

New Environment

The X Window System is relatively easy to comprehend. The graphic user interface (hereafter referred to as GUI) offers programmers a conducive development atmosphere for programming. However, to master the Oracle - X Window System, a programmer should be allotted time resources to the GUIs and the system-level processes of the X Window System.

The first task is to become familiar with the X clients. The clients in the core distribution from MIT perform most of the system and application administration between both development environments. (Some of the X Window System clients will be outlined in this paper.) Another task is to execute previous applications already developed on the X terminal. By testing previous applications, programmers can modify past applications to suit the X Window System and be able to anticipate future presentation displays for upcoming projects.

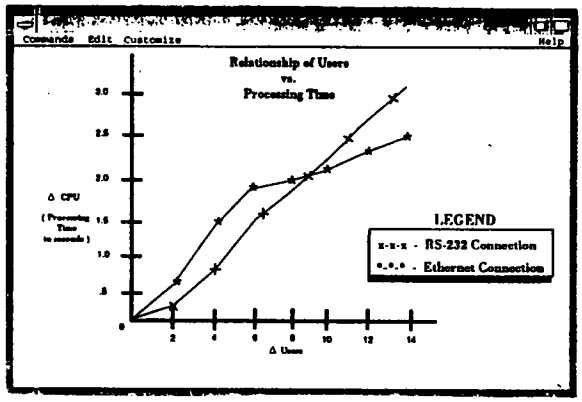
The application programmer with the bit-mapped terminal must be allowed to explore the nuances of his or her new environment. Allowing programmers to work on the X terminals will consume less development time and will decrease run-time errors due to unfamiliarity.

Application Behavior Between Environments

The overall appearance of SQL*Forms 3.0 in the X window is very similar to the character-based presentation, but when testing commenced, the differences became clear. The SQL*Forms applications in the X Window System environment exhibited a lethargic cursor movement during field-to-field consistency testing. Data are captured by 'SELECT' statements in the 'KEY-NXTFLD' trigger in the field-to-field testing methodology. Field- to-field methodology causes increased network traffic, because 'SELECT' statements are executed at every field for database validation. Also, greater resources are being utilized for the X terminals, because of the competition for CPU time from other X terminals on the host. If a SQL*Forms application contains 20 fields in a screen, the field-to-field 'SELECT' statements must cluster data to validate on each field.

Because the character-based terminals use RS-232 cable, there is a direct hookup to the host. The X terminals are connected to a DEC Server200, via twisted-pair ethernet connection, then to the X host. During testing, as more users utilize the X host, the ethernet topology displayed greater performance degradation than the RS-232 topology. This performance degradation is due to the twisted-pair ethernet connection to the host via the DEC Server200. Despite this performance degradation, the client- server architecture has distinct advantages over the RS-232 architecture. Multiple RS-232 users displayed a performance decrease when executing SQL*Forms applications. In the same test, the client-server architecture illustrated that X terminal's CPU and the host's CPU work together to process the data and to redraw the screen faster than character-based terminals (see Figure 1).

Figure 1



The pop-up screens are defined in the Image/Modify/Page Definition Screen within SQL*Forms 3.0. The X terminals and character-based terminals define the X and Y coordinates for the location of the pop-up screen. There is a slight difference between the presentation of X and Y coordinates between each of the respective environments.

The 'CALL(NO_HIDE)' Oracle function operates similar to the pop-up screens mentioned in the previous paragraph. The host SQL*Forms application utilize the 'CALL(NO_HIDE)' function to display a guest SQL*Forms application over the host SQL*Forms application. The design phase specified the 'CALL(NO_HIDE)' function would place

the guest SQL*Forms in the upper left-hand corner of the host SQL*Forms application. In the development phase, the programmers used the X terminals for the location of the guest SQL*Forms application. In the testing phase, a common occurrence was the misplacement of the guest SQL*Forms application within the host SQL*Forms application.

While working with SQL*Forms 3.0 in the DECterm (DECterm, a DEC X client, can emulate 10 different VT modes; we used VT100 mode), the programmers had difficulty locating the cursor within the application. The SQL*Forms application text was black, and the background was white. The cursor, being the same color as the fields, made it difficult to find the cursor on the screen. For the best results, change the cursor attributes in the resource manager, so that the cursor will blink. Although, the cursor will still be difficult to view, it will be easier to find than a non- blinking cursor on a field.

Pro*C with X Window System

At the IRS, Pro*C applications were also developed for use with the X Window System. This section will focus on that experience.

When developing Pro*C applications, the programmers were working in the X Window System environment. Since our end-users are using character-based terminals, there were some anomalies in the batch programs. The relationship between pixels and columns are not consistent between their respective environments in output file presentation. The programmers used the xterm program that is designed to be a terminal emulator for the X Window System. The xterm function provides VT102 compatibility for the programs that can not use the X Window System directly. Even the xterm window, upon viewing the file, was not adequate for the programmers. The screen representation is important for the developers to be aware of the phenomenon and to thoroughly test the output files on a character-based terminal.

A customer wanted a Pro*C application to produce an output file that required the character-based terminal to emulate 132 column character-based mode. The SQL*Forms 3.0 application can execute a command to change the terminal emulation from 80 column to 132 column mode. Upon selection of the batch file in the SQL*Forms application, the window was altered by using the command to change to the terminal mode. The 132 column mode made testing the file impossible on an X terminal. Instead, testing of this application had to be done on a character-based terminal, in order to meet user specifications.

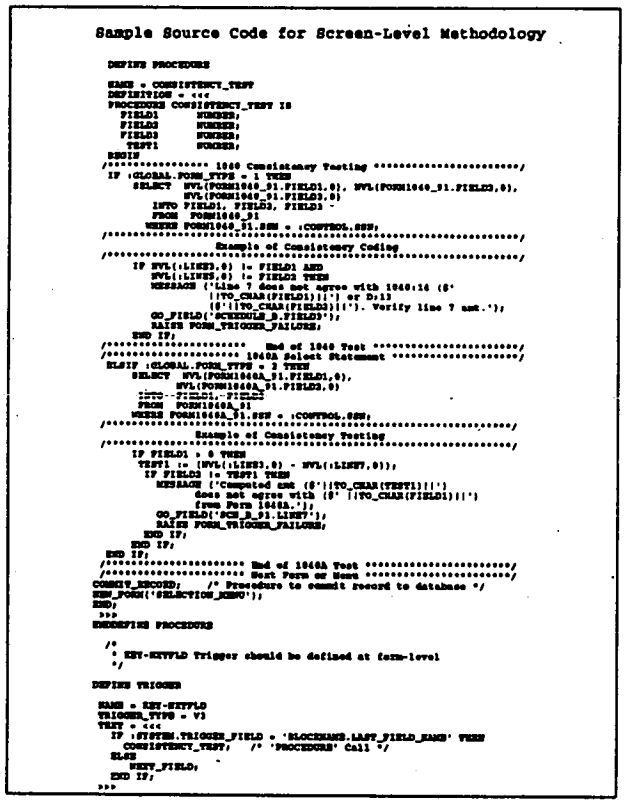
Pro*C programmers must be aware of the signal handlers in the UNIX operating system. A UNIX signal handler may interfere with the X protocol for communicating with the X server. The X protocol is responsible for the communication between the host and the client. If the UNIX signal appears in mid-stream of the X protocol, the result could be the destruction of the window running the Oracle application. Normally, Oracle programmers do not have to be concerned with the X protocol, because it is too low-level and completely transparent. However, programmers should be aware of the background processes, in case the signal-handler situation arises.

Tuning the Oracle - X Window System Environment

In testing the X Window System for our own applications, we noticed a few problems that may arise when using the Oracle - X Window System. This section will focus on those issues and describe the refinements we have introduced to deal with them.

The irregular movement of the cursor during field-to-field testing was alleviated by screen-level testing methodology. The major advantage of screen-level methodology is a reduction in the number of 'SELECT' statements to the Oracle database. The consistency testing was executed in a 'PROCEDURE' function, defined in the form to be executed at the last field in the screen. Within the 'PROCEDURE' function, one 'SELECT' statement would retrieve data to the SQL*Forms application. If an error is encountered, a 'GO_FIELD' function would send a cursor to the appropriate field and execute 'RAISE FORM_TRIGGER_FAILURE' (see Figure 2). This methodology allows for faster consistency testing on X terminals, as well as character-based terminals. During tests, the application gave the appearance of executing faster as the number of users increased on the host. Actually, the appearance of executing more rapidly has do to the X terminal's internal RAM ability to redraw the screen faster than the character-based terminal.

Figure 2



```
Sample Source Code for Screen-Level Methodology

 DEFINE PROCEDURE

 NAME = CONSISTENCY_TEST
 DEFINITION = <<<
 PROCEDURE CONSISTENCY_TEST IS
   FIELD1      NUMBER;
   FIELD2      NUMBER;
   FIELD3      NUMBER;
   TEST1       NUMBER;
 BEGIN
/*************** 1040 Consistency Testing ********************/
 IF :GLOBAL.FORM_TYPE = 1 THEN
    SELECT  NVL(FORM1040_S1.FIELD1,0), NVL(FORM1040_S1.FIELD2,0),
            NVL(FORM1040_S1.FIELD3,0)
      INTO FIELD1, FIELD2, FIELD3 -
      FROM  FORM1040_S1
     WHERE FORM1040_S1.SSN = :CONTROL.SSN;
/*************************************************************/
                    Example of Consistency Coding
/*************************************************************/
    IF NVL(:LINE3,0) != FIELD1 AND
       NVL(:LINE5,0) != FIELD2 THEN
       MESSAGE ('Line 7 does not agree with 1040:1& (&'
                ||TO_CHAR(FIELD1)||')' or D:13
                (&'||TO_CHAR(FIELD2)||')'). Verify line 7 amt.');
       GO_FIELD('SCHEDULE_A.FIELD3');
       RAISE FORM_TRIGGER_FAILURE;
    END IF;
/*********************  End of 1040 Test *********************/
/*********************  1040A Select Statement ***************/
 ELSIF :GLOBAL.FORM_TYPE = 2 THEN
    SELECT  NVL(FORM1040A_S1.FIELD1,0),
            NVL(FORM1040A_S1.FIELD2,0)
      :----:FIELD1, FIELD2
      FROM  FORM1040A_S1
     WHERE FORM1040A_S1.SSN = :CONTROL.SSN;
/*************************************************************/
                    Example of Consistency Testing
/*************************************************************/
    IF FIELD1 > 0 THEN
       TEST1 := (NVL(:LINE3,0) - NVL(:LINE7,0));
       IF FIELD3 != TEST1 THEN
          MESSAGE ('Computed amt (&'||TO_CHAR(TEST1)||')
                   does not agree with (&' ||TO_CHAR(FIELD1)||')')
                   (from Form 1040A.');
          GO_FIELD('SCH_B_S1.LINE7');
          RAISE FORM_TRIGGER_FAILURE;
       END IF;
    END IF;
 END IF;
/********************* End of 1040A Test *********************/
/********************* Next Form or Exit *********************/
COMMIT_RECORD;    /* Procedure to commit record to database */
NEW_FORM('SELECTION_MENU');
END;
 >>>
ENDDEFINE PROCEDURE

  /*
  * KEY-KEYFLD Trigger should be defined at form-level
  */

DEFINE TRIGGER

 NAME = KEY-KEYFLD
 TRIGGER_TYPE = V3
 TEXT = <<<
   IF :SYSTEM.TRIGGER_FIELD = 'BLOCKNAME.LAST_FIELD_NAME' THEN
      CONSISTENCY_TEST;   /* 'PROCEDURE' Call */
   ELSE
      NEXT_FIELD;
   END IF;
 >>>
```

A Berkeley System Development (BSD) function iostat prints a number of I/O statistics that will assist the programmer with Oracle and UNIX system performance. The iostat function can use the following flags '-c' or '-t.' The '-t' displays the percentage of time each CPU spent in user mode, running low priority processes in system mode and idling. The '-c' option continuously updates the monitoring of the system in one second intervals to be specified by an integer. Since the iostat function is device-specific, a user can also retrieve I/O statistics on the X terminal. Utilizing two windows, execute the following command in one of the windows as follows:

```
iostat -c  X terminal name  integer
```

In the other window, execute SQL*Forms applications while monitoring the network usage of the desired X terminal. The display will show all disk drives for report. Below each disk drive are 'bps' and 'tps.' The 'bps' is the average number of kilobytes per second during the previous interval. The 'tps' indicates the average number of transfers per second during the previous interval. The iostat function is a very useful tool to determine the amount of CPU usage with each 'SELECT' statement between field-to-field and screen-level methodologies.

Memory allocation is important because the X server requires a tremendous amount of resources for the X clients. Modulating the SQL*Forms applications is recommended for form-to-form control methodology. The 'NEW_FORM' Oracle function terminates the current SQL*Forms application and enters the next SQL*Forms application. Using the 'NEW_FORM' function on large projects alleviates the need for large memory resources, because the function will clear the current SQL*Forms application from memory and access the desired SQL*Forms application. Information within the SQL*Forms can be transferred by initializing 'GLOBALs' within the SQL*Forms application. In other words, the 'NEW_FORM' methodology relieves some of the laborious usage of the random access memory having to 'swap in' a number of SQL*Forms applications while executing X clients on the system.

In a particular environment, when the developers are using X terminals and end-users have character-based terminals, it is beneficial to remove the '\t's from the Pro*C source file. The '\t' is a horizontal tab display command used in output functions. The '\t's are displayed differently when viewing the output file in its respective environments. By removing the '\t's and using only character spaces to denote space between words, allows changes to be made quickly when testing Pro*C files between environments.

Within the Ultrix header file (/usr/sys/h/param.h), the system parameter NCLIST can be adjusted for better performance. The NCLIST parameter is the number of clist segments. A clist segment is 12 characters. These characters have arrived from a terminal and are waiting to be given a process number. Thus, enough space should be allocated so that every terminal can have at least one average line pending (about 30 to 40 characters). The parameter should be set for multiples of 2 for each X terminal within the network.

The xload function provided by MIT in the core distribution is important for day-to-day use on the host. The xload function ascertains system us-

age and converts the information to a histogram GUI display. If a host system is being inundated by users, a programmer can access another host on the network and retrieve information. In addition, since the X Window System is network-based and able to execute multiple clients on different hosts, xload icons on the display can represent each host within the network system. The user can monitor each host within the network to observe application performance on different hosts.

To assist the programmers operating on several hosts, it is advisable to customize the prompt, in order to save time for the programmers. Modify the set prompt in the .login file:

```
set prompt = "^[[[m^[[5m^[[7m 'hostname' (\!) - "
```

Setting the prompt to display the hostname prevents the programmer from being confused by having a multitude of displayed windows on different hosts.

NCD terminals, like most other terminals, have an Oracle keyboard mapping resource file in the /oracle/forms30/admin/resource directory called xtermncd.r. Adapting the resource file is quick and easy; however, it is not the only solution. The xmodmap function in the core distribution, allows the programmer to quickly adapt the X terminal keyboard to a character-based keyboard mapping. The resource file remains concurrent when executing 'runform' or 'iapx30' commands for programmers and end- users. To examine the keyboard mapping, execute the following command: 'xmodmap -pk keyboardmapping.doc'. By executing this command, the xmodmap function will create a file to assist the programmer designating Oracle function keys. The xmodmap function will cut down a significant amount of incompatibilities between developers and end-users, resulting in different resource files being utilized during project development.

The xcutsel function in the MIT core distribution allows the programmer to cut and paste routines. In the DEC terminal representation in X window, a text editor (vi) is used to edit the SQL*Forms application, as follows:

- Place the pointer in the upper left-hand corner and press the left button to highlight the text to copy to the other window.
- After the highlighting, go to the xcutsel menu and click the 'copy primary to 0.'
- Activate the desired window for text copying, move the cursor to the desired location within the target window, and press the 'copy 0 to primary' in the xcutsel window.
- Move the pointer to the target window location and press the middle button on the mouse to place the highlighted text.

The xcutsel function will assist programmers developing applications that standardize 'PROCEDUREs' and 'TRIGGERs' between applications.

During development of the SQL*Forms applications, the pop-up screens were created on the X terminal. The specifications instructed that a pop-up window should be displayed in the upper left-handed corner of the display. The solution is in the Painter/Modify/Page Definition Menu, which allows the programmer to specify the X and Y coordinates of the page location. For best results, the pop-up window must be defined in the middle of the page in the window. The character-based terminal will displace the pop-up screen by two columns above the designated Y coordinate.

The 'CALL(NO_HIDE)' function locates the guest SQL*Forms within the host SQL*Forms application. In the development phase, problems were never encountered. When the character-based terminals were used, the project required the guest SQL*Forms application to be placed in the upper left-hand corner of the character terminal. The 'ANCHOR_VIEW()' function in the guest SQL*Forms 'KEY-STARTUP' locates the SQL*Forms in a declared position onto the screen. An example is below:

```
  ANCHOR_VIEW(1,TO_NUMBER(:GLOBAL.X),
TO_NUMBER(:GLOBAL.Y));.
```

The global XY coordinates can be initialized in the host SQL*Forms application before the 'CALL(NO_HIDE)' function or in the host 'KEY-STARTUP' trigger. Using the 'KEY-STARTUP' trigger for setting the XY global coordinates is the most favorable way to standardize the display throughout the project.

Finally, when system upgrades are performed, it is advisable to have the database administrator, or the systems analyst, produce a full listing of the directories on the X server host. When upgrades to the operating system are done, a directory may not be included during the system modification. The 'ls -R directories.doc' com-

mand, when executed in the root directory, will list all the directories on the host. If there are any problems encountered with the X Window System at startup, the database administrator can refer to the list to notice any dissimilarities between the printouts.

Future Considerations

A number of operating systems and database management systems have been created without using non-intuitive user interfaces. In the future, users will have standard 'Point-and-Click' ability to retrieve data within a highly graphical-interactive system. The 'Point-and-Click' environment is more inviting than traditional character-based terminals because of increased production by using GUIs.

The X Window System, or any other windowing platform, can greatly benefit from the use of a client function that lets the programmer or end-user 'page through' SQL*Forms applications. This client would be programmer-defined to allow a user to change a SQL*Forms application by using a mouse to click 'page-forward' or 'page- backward.' Having a 'Point-and-Click' ability permits the user to 'turn pages' of SQL*Forms applications. The client would have the ability to check for any mandatory fields in the displayed SQL*Forms application. If a NOT NULL field is encountered, the cursor would go to the required field and execute a 'RAISE FORM_TRIGGER_FAILURE.' The simple Oracle/X client function would increase the familiarity of GUIs and would be a productive tool for programmers and users.

A derivation of the xload client in the core distribution would be a program that monitors Oracle kernel usage. The function would display information in a histogram format that the user can place on the X desktop environment (see Figure 3). The client has the benefit of being able to monitoring multiple Oracle databases for the programmer or administrator.

As end-users utilize the X Window System more, a package function would be useful to create a MOTIF window from another window. The propagation of the other MOTIF window would contain a SQL*Forms application (see Figure 4). This function would have the same qualities as an existing Oracle function 'CALL(),' but will be able to create another child X window on the desktop screen (see Figure 5). The function would have several advantages: the application would have a 'Look-and-Feel' appearance for user-friendliness and the programmers could allocate more time for database design, instead of learning Xlib.h Programming.

Figure 3

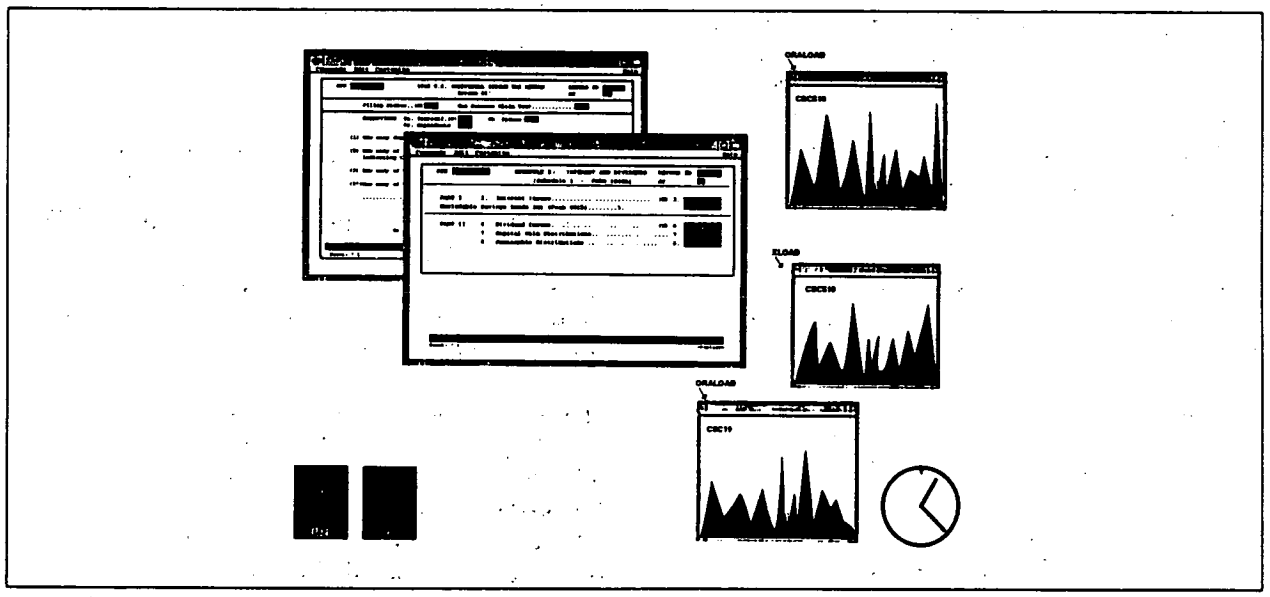


Figure 4

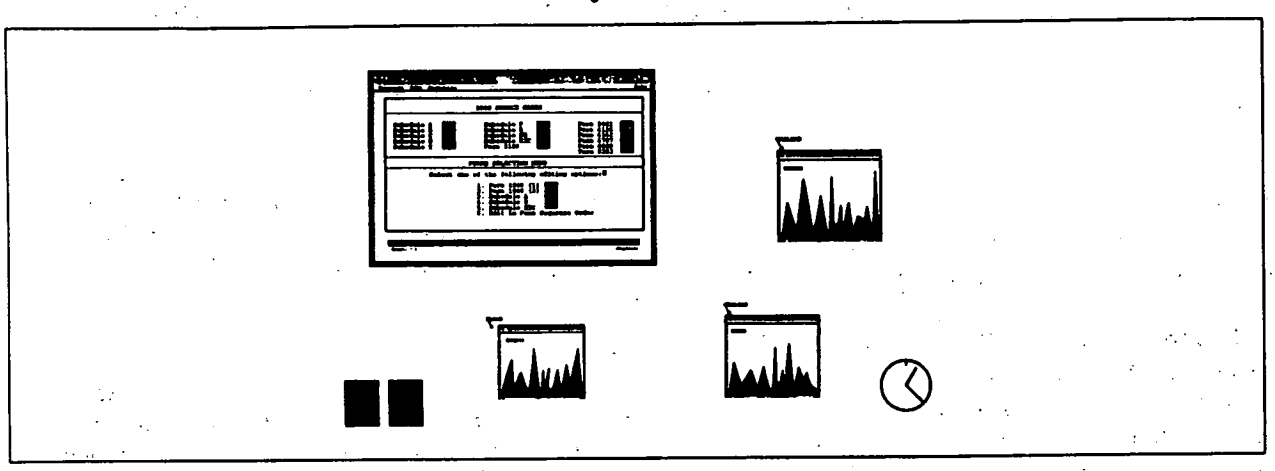
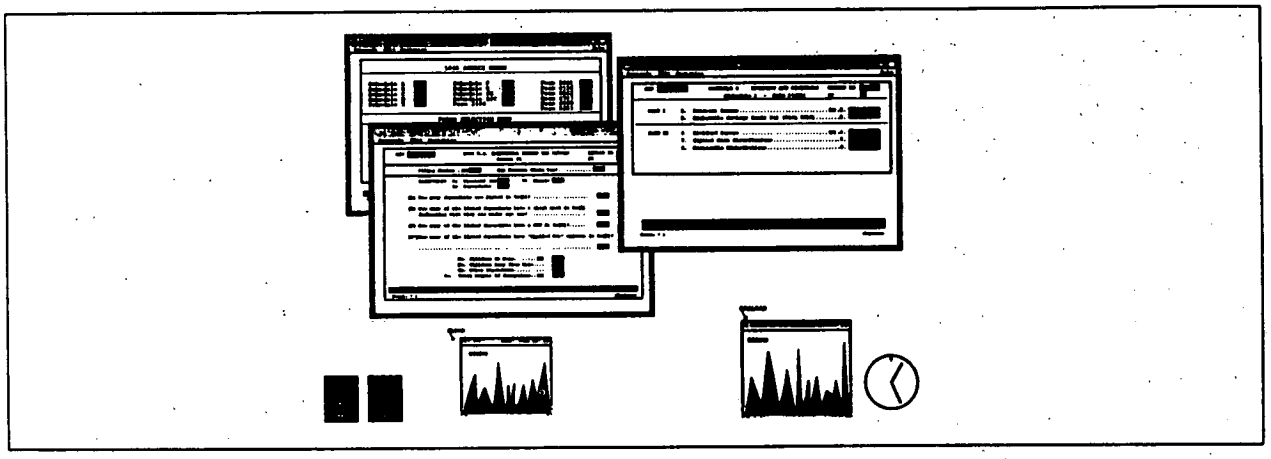


Figure 5



Conclusion

Recently, 75% of FORTUNE 1000 Chief Information Officers (CIOs) indicated that graphic user interfaces have already been incorporated into their companies' computer systems, or will be utilized in the next five years. Oracle Corporation is one of the market leaders in the area of relational database technology. As the X Window market emerges as the norm for windowing environments on platforms, the computer community should expect an integration of data accessibility and user-friendliness.

Software programmers and vendors are required to anticipate the needs of tomorrow's user. The rapid success of various windows platforms solidifies the premise that users want an intuitive 'Look-and-Feel' program. To be competitive in a global market, designers and programmers must create applications to suit client's needs and expectations.

Appendix

The applications discussed in this paper were developed on the following platforms:

Minicomputer - Digital Equipment Corporation 5810

Minicomputer - Digital Equipment Corporation 5100

- MIT X Window System X11 / Release 4.0.
- NCD X Server Software 2.3.0. (OSF/MOTIF Look-Alike)

Digital Equipment Corporation and Oracle Corporation software include:

- Ultrix 4.2 BSD with C compiler
- ORACLE RDBMS 6.0.33.1.1
- Oracle SQL*Forms 3.0.16.1
- Oracle SQL*Plus 3.0.9.1.2
- Oracle PL/SQL 1.00.32.03.01