An Interactive Display Environment, or knitting sheep’s clothing for a wolf

by ROBERT P. O’HARA

The IBM Corporation

Yorktown Heights, New York

ABSTRACT

AIDE, An Interactive Display Environment, attempts to improve the usability of a traditional timesharing system by exploiting the use of a powerful personal computer with a large, high-resolution all-points-addressable (APA) display as a terminal. No changes to the timesharing system are required, and existing application programs dependent on alphanumeric display terminals run unchanged. A sample terminal session illustrates AIDE in use. Ways in which the timesharing system might be modified to further exploit the display are suggested.
MOTIVATION

The display interface of an interactive system represents a major part of the face the system presents to the user. Most traditional timesharing systems such as VM/370 CMS\textsuperscript{11} (VM/CMS) and UNIX\textsuperscript{10} were designed with typewriter terminals as the assumed user terminal. In such systems the user types commands as strings of words; the system types back a response that may be many lines long. This conversation is recorded on the typewriter terminal’s paper. If the user wishes to review an earlier portion of the conversation, it is readily available.

Display terminals (hereafter called displays) have supplanted typewriter terminals as the standard interactive terminal. They offer advantages such as very high speed operation and the ability to present a formatted display. Programs that use these formatted full-screen displays, such as visual editors and menu-driven applications, offer increased ease of use and productivity over the previous typewriter-mode programs, but typically fit poorly with the rest of the system. The user switches between typewriter mode and full-screen mode during the interactive session, losing whatever was on the screen during a mode switch. The typewriter-mode interface at a display is essentially unchanged from that at a typewriter terminal. The conversation that previously was recorded on the typewriter paper now scrolls off the screen and is lost, as if someone came along and tore off the paper every 24 lines. In other words, a valuable function (that of review of previous commands and responses) has been lost.

Several window/terminal management systems that provide more than “typewriter mode on a display” have been implemented. Notable for support of alphanumeric displays are the IBM TSO Session Manager,\textsuperscript{7} the RIG Virtual Terminal Management System at the University of Rochester,\textsuperscript{8} BRUWIN at Brown University,\textsuperscript{9} the Automated Desk,\textsuperscript{14} and Hartman’s TOY system.\textsuperscript{3}

I believe that large-screen all-points-addressable (APA) displays are as important an advance in function and usability as alphanumeric displays were over typewriter terminals. Far more information can be displayed on them, and their APA capability allows the use of multiple fonts, graphics, and images. Systems such as Smalltalk\textsuperscript{13} and the Xerox Star\textsuperscript{8} have been designed with APA displays in mind. These systems are visually oriented, designed around a pointing device such as a mouse. They do not offer either a traditional command language interface or a “fill in the blanks” menu interface, but rather they define new paradigms for the user-system interface, based on the capabilities of an APA display and a powerful personal computer.

In this project I chose to explore how one can use such displays with an existing interactive system. It is often not practical to rewrite the system to support an APA display. Existing programs, many with dependencies on alphanumeric displays, must continue to operate.

GOALS

The goals of the project were to

- enhance the usability and function of the system by fully exploiting the APA display.
- provide a base for an evolution to new modes of display usage and user interfaces.
- allow all existing programs of the interactive system to run unchanged, including “full-screen” programs.
- not modify the existing interactive system.

GUIDING PRINCIPLES

So how does one go about designing a better user interface? The approach used in developing AIDE was to develop a list of principles to guide the design. The intent was to have these principles embody good human factors, and thus assist in achieving that better interface. The design principles were the following:

- Preserve the context of the user’s conversation with the system. This is important because the relationships among commands entered during a session are often as important as the commands themselves; there is continuity to the user/system conversation.

  \textit{Show more of it.} The typical display terminal has 24 lines of 80 characters. Note that this is not even one page of text. When the screen is split into two or more windows, very little information can be displayed in each window. It’s difficult to be productive when one views the world through a keyhole. Large-screen displays such as the one used by AIDE can display over 90 rows of 100 characters, and even this is not too big!

  \textit{Avoid its loss.} Most systems clear the screen when entering and leaving the editor. It is not uncommon to observe users copying information from the screen with pencil and paper before it is lost. AIDE creates a separate window for the editor or other full-screen programs, preserving the information displayed in the typewriter-mode window.

  \textit{Avoid its disruption.} Asynchronous messages from other users and programs are typically displayed immediately on the screen, interspersed with and interrupting
whatever else is there. If the user is within an editor, the editing session is typically interrupted, the screen cleared, and the message displayed. Before resuming the editing session, the user must copy the message on paper if it is not to be lost. AIDE places asynchronous messages in a separate window to avoid disrupting the current context.

- Provide feedback to user actions. Norman also emphasizes the importance of feedback in preventing many classes of user errors. AIDE provides instant feedback within pop-up menus by using reverse video to highlight the command point at by the user. Smooth scrolling of the message window one raster line at a time gives the user instant feedback and easy control of the amount of scrolling desired. It seems to eliminate the problem of windowing versus scrolling common on many systems.

- Don’t hide the state of the system. Norman also observes that “mode errors occur when the person believes the system is in one state (mode), whereas it is actually in another.” AIDE indicates the modes of the system in several different ways:

1. Different cursor shapes are used to indicate the different states of the system and terminal:
   - Insert/replace mode of the character cursor. AIDE changes the shape of the character cursor itself, from an underscore (replace mode) to a reverse-video box (insert mode).
   - Typewriter/fullscreen mode. The pointing cursor on the screen changes shape when fullscreen mode is entered, indicating that it may now be used to move the character cursor within the fullscreen window.
   - Screen manipulation menus. A different pointing cursor appears when the screen manipulation menus are active. This informs the user that the pointing cursor is now to be used to make a selection from the menu.

2. Separate windows are used for alphanumeric fullscreen programs. Displaying the fullscreen editor in a separate window superimposed on the typewriter-mode window clearly indicates that the user has entered a different environment. Burying the fullscreen window upon leaving the editor clearly shows the return to the system level.

3. Recursion levels in the system, such as within the editor, are shown by surfacing additional windows, with the current level of recursion displayed in the topmost window. With a minor change to the editor an additional fullscreen window is created when additional files are edited. This has been a very useful approach with VM/CMS, for many of the most used programs such as electronic-mail sending and reading, file-list manipulations, system help information and so on are based on the editor.

- Eliminate modes where possible. The typewriter mode of VM/CMS at a display terminal presents six different states to the user. The rules governing which key to press when are very complex. AIDE simplifies life at the terminal by handling typewriter mode for the user. For example, the CLEAR key is pressed when needed.

IMPLEMENTATION OVERVIEW

Connecting non-IBM equipment to IBM computers is often a challenge; connecting the personal computer on which AIDE is implemented to the IBM host system proved no exception.

A Three Rivers Perq computer is the terminal used by AIDE. The Perq has a specially microcoded processor that executes PASCAL intermediate code at up to 1 million instructions per second and a 24-megabyte hard disk. A 768 x 1024 100 points-per-inch black and white APA display directly displays 96k of the 1 megabyte of memory. A Rasterop instruction copies any size area of memory, including the memory displayed on the screen, to any other area of memory in one refresh cycle of the display. This allows for very fast screen operations. The Perq also has a keyboard and tablet with a pointing device. The pointing device is a Summagraphics Bit Pad tablet and puck. The Perq connects to an IBM Series/1 computer over a 9600 baud full-duplex asynchronous line. The Series/1 connects to an IBM S/370 model 3031 AP computer through a block-multiplex channel.

Software

AIDE is written in PASCAL and runs on the Perq. It uses a window management system called JAWS (Jaws A Window System) that is also written in PASCAL. In fact, all of the software running on the Perq is written in PASCAL, including the operating system, since the Perq is microcoded to efficiently execute PASCAL intermediate code.

AIDE makes the Perq appear as an IBM 3101 ascii teletype display terminal, which is one of the terminals supported by the IBM Series/1 Yale Ascii Terminal Communications System Installed User Program (the Yale IUP, program number 5796-RBT) running on the Series/1. The Yale IUP in turn makes the IBM 3101 look like an IBM 3277 display terminal to the System/370 model 3031 computer. The 3031 is running the VM/CMS operating system, and it is to CMS that the user finally logs on.

Operation

The result of all this deception is that CMS believes the user is at a locally attached IBM 3277 alphanumeric display terminal. AIDE views the terminal screen as a 24 by 80 character array. When the terminal is in typewriter mode (line at a time operation) AIDE interprets this screen and displays user input, system output, and messages in various windows, as illustrated in the sample terminal session below. When a fullscreen program is invoked, AIDE recognizes this and creates a 24 by 80 screen and program function (PF) key pad.

AIDE also has a small repertoire of actions that can be invoked from programs running on the host computer. These actions include creating and hiding fullscreen and line-mode windows.

SAMPLE TERMINAL SESSION

This sample terminal session illustrates the facilities provided by AIDE. It is a series of snapshots of the display screen.
Unfortunately the printer used does not copy the right half-inch of the display screen, so some clipping of the images has occurred.

Figure 1 shows the terminal screen shortly after the user has logged on to VM/CMS. There are three windows visible on a restful gray background. In the upper right-hand corner is a window that displays the current date and time. The largest window is titled “VM/CMS Console Log.” It displays user input and system output of the session while in typewriter mode. It is analogous to the roll of paper in a typewriter terminal.

When the window fills with data, the data are scrolled up to allow new lines to be displayed. The window at the bottom of the screen is titled “VM/CMS Console Input.” The character cursor is displayed in this window; here the user enters commands and other input. In the lower right-hand corner of this window a status message is displayed. This status message is generated by VM/CMS, not by AIDE.

The console input window is simply viewing the bottom two lines of the simulated display terminal, which is the command entry area for VM/CMS.

Near the center of the screen is a small arrow pointing to the upper left. This is the *pointing cursor* that follows the position of the pointing device on the Perq’s tablet. Its image on the screen is maintained by special hardware in the Perq.

The other user has responded to the initial message of “howdy” by sending several messages of his own. These are displayed in a separate window titled “External Messages.” This window surfaces automatically whenever a new message is received from another user or system. VM/CMS prefixes each message with a timestamp and its origin.

Since this window overlies the console window, the user may wish to *bury* it beneath the console window. To do this the pointing device is used to move the cursor (pictured as an arrow on the screen) so that it points at the message window. Then one of the buttons on the pointing device is pressed.

Instantly, a *pop-up menu* appears (Figure 2) at the position of the cursor. The cursor’s shape is changed to signal that a selection is to be made from the menu. As the cursor is moved over the menu, the commands are displayed in reverse video, providing the user with instant feedback as to what command will be issued. Since the user wishes to bury the message window, Bury is selected by pressing the pointing device. The menu disappears and the window is buried beneath all other windows. Its location on the screen has not changed. Had the user chosen instead not to select any of the commands, pressing the pointing device while the cursor is outside of the menu would remove it from the screen without executing any commands.

In Figure 3 the user has issued the `rl` command (a synonym for RDRLIST) which lists the files in the user’s mailbox (virtual card reader). This command uses the system editor to edit the list of files in a fullscreen manner. AIDE recognizes that a fullscreen program has been invoked and in response it creates a window the size of the simulated terminal display screen (24 by 80 characters) and a program function key pad with the 12 PF keys arranged as they are on an IBM 3277 terminal. The typewriter-mode input window has disappeared because it is not useful while in the editor. The cursor’s shape has changed to indicate that a fullscreen program is active.

The pointing device can be used to directly position the character cursor on the simulated 3277 screen, avoiding the use of the traditional cursor movement keys. Positioning the character cursor in this manner is much faster than holding a key and waiting for the cursor to crawl across the screen. However, for small cursor movements of a couple of characters or so the cursor movement keys on the Perq keyboard are faster than the pointing device. This is especially true if other control keys such as insert and delete are being used.

The same pointing device is used to press the PF keys displayed on the screen: the pointing cursor is moved to the PF key and pressed. Thus PF key oriented programs like RDRLIST become very easy to use. To view a file in the list, the user points at the file name and presses to move the character cursor, then points at PF 11 and presses to press the PF key.

The PEEK command thus invoked calls the editor recursively. With a small change to the profile of the PEEK command AIDE is requested to create another fullscreen window, visualizing the recursive call to the user. The first fullscreen window is now dormant until the PEEK command returns to it. The user has decided to reply to the note displayed by PEEK. From the PEEK screen the NOTE command is issued (Figure 4), which invokes the editor in a recursive manner. AIDE again creates another fullscreen window.

Looking at Figure 4, the reader might conclude that the display screen appears rather cluttered with windows, and that AIDE has made the terminal environment more confusing than before. For the user at the terminal, this does not appear as a problem. The windows appear in direct response to user actions, under control of the user, over time as the session proceeds. An analogy to working at a desk can be drawn. During the course of the day, a person places papers and opens books on the desk. Someone else viewing the desk might see a clutter, but to the person at the desk it is manageable and useful to have many pieces of information partially visible at once.

It is important to contrast the screen image shown here with that of a standard display terminal. With AIDE, the user sees exactly where he or she is in the interactive session. At a standard terminal the user would view only the current screen image, and the contents of the screen prior to entering the fullscreen program, including all external messages, would have been lost.

Later in the session the user displays the file sent from the colleague he or she earlier conversed with. Wishing to review that conversation, the user points at the buried message window and presses the pointing device. The menu pops up and the user points at the Surface command.

Pressing the pointing device again causes the window to surface instantly (Figure 5). To scroll back to view the previous messages, the user points to the scroll arrow on the right side of the window. Pressing the pointing device causes the window to scroll smoothly one raster line at a time over the data. The speed of scrolling is related to where on the scroll arrow the user points. The scroll arrow is located on the right-hand side of the message window. Close to the center the rate is slow; at the heads it is quite fast. The smooth
Figure 1—Shortly after logon to VM/CMS
Figure 2—The message window and pop-up menu
Figure 4—The Note command invokes the editor recursively.
Figure 5—Smooth scrolling of the message window
scrolling allows the data in the window to be read while it is being scrolled, and gives instant feedback as to the direction of scrolling.

CONCLUSIONS

Unfortunately, my sabbatical assignment ended after AIDE had been operationally stable for little over a month. During that time I used AIDE regularly at my desk to access VM/CMS. (I even gave up my regular alphanumeric display terminal!) Based on that short experience the following thoughts are offered.

It is practical to build an improved user interface as a “front end” to an existing interactive system without modifying that system. Such an interface can run existing alphanumerical display application programs and yet provide a superior user environment.

The large screen is wonderful. In comparison, a standard 24 by 80 display terminal is like trying to view the world through a keyhole.

The font used in AIDE can display more than 90 lines of more than 100 columns of characters on the screen. Yet even this is not large enough. As can be seen in the sample terminal session, the display can become crowded with windows.

Multiple overlapping windows are an effective way to organize the information displayed on a large screen. Displaying additional windows instead of clearing the screen seems to reduce confusion about where one is. Since the screen is never cleared, as so often it would on a standard display terminal, the need for copying information off of the screen onto paper is greatly reduced.

Pointing devices are a very natural way to select from menus or position a cursor. The use of pointing devices and instantaneous pop-up menus can reduce the need for many commands and modes that are common in most systems.

Areas for Further Study

There are many possible areas for additional work. The following items were planned for AIDE but were not implemented owing to lack of time.

Add additional functions to the windows. It would be nice to shrink windows that are not of current interest to make them occupy less of the display screen. Windows that are inactive might be indicated as such by shading them. Typewriter-mode windows might copy their contents to disk so that a complete session log could be retained.

Provide soft program function keys for 3277-based programs. With little change to most programs, the names of the functions performed by pressing the PF keys could be sent to the Perq, and AIDE could thus dynamically label the PF keys. The PF key pad could be displayed and operated as a pop-up menu, thus providing more immediate feedback to the user.

Provide separate windows for system status functions such as mailbox files, terminal and system status settings, current disk links, directory settings, and search paths. These windows would always show the current status of their functions, and a setting could be changed by directly altering the window. This approach is similar to the property and option sheets of the Xerox Star.

Provide graphic displays of system information. Most systems display information such as load, utilization, and search orders as tabular lists. The graphic capabilities of the APA display could be easily exploited to provide this information in a more comprehensible form such as bar charts for load information and tree graphs for directory structures.

ACKNOWLEDGMENTS

This work was done while I was on assignment at the IBM Cambridge Scientific Center in Cambridge, Massachusetts. I would like to gratefully acknowledge Dick MacKinnon for inviting me to the IBM Cambridge Scientific Center, my manager Alex Chandra and my director Dick Kelisky for sponsoring my sabbatical assignment. I also thank my colleagues at the Scientific Center for their advice and support during my stay, especially Sheldon Borkin, John Prager, John Gonzalez, John Webster, Ray Fessel, and Peter Hardy.

REFERENCES
