HUMAN-COMPUTER INTERACTION RESEARCH AT THE UNIVERSITY OF MARYLAND

Ben Shneiderman
Department of Computer Science and
Human-Computer Interaction Laboratory
University of Maryland
College Park, MD 20742

October 1985

The Human-Computer Interaction Laboratory (HCIL) is a unit of the Center for Automation Research at the University of Maryland. HCIL is an interdisciplinary research group whose participants are faculty in the Departments of Computer Science and Psychology and the Colleges of Library and Information Services, Business, and Education. In addition, staff scientists, graduate students, and undergraduates contribute to this small, but lively community that pursues empirical studies of people using computers.

Our support comes from industrial research projects, government grants, the State of Maryland, and the University of Maryland. Projects often become interrelated in surprising ways enabling individuals to cooperate constructively. Some of our efforts during the past year are described below.

1) Programmer workstations

We have been exploring ways to attach three or more separate screens to an IBM PC and use each screen as a window. This gives a full 25/80 display space for each window. We also have used a single very large display, the IBM 3290 Plasma Display Workstation with 62 lines of 166 characters, to get the large window sizes we want.

We have conducted observational studies and built many variations of multiple screen formats under support from IBM Federal Systems Division in Bethesda, Maryland. These studies have led us to the following concepts:

Fusion: Several screens are 'fused' together at the bottom/top or right/left edge to form one, continuous logical screen. Fusing two screens at the bottom/top gives a 50 line, 80 column display. Fusing two screens at the right/left edge gives a 25 line, 160 column display. We can display whole pages of code at once or all of a spreadsheet.

Copying: One central screen is used like current single screens. Several other screens are used to display copies of text copied from the central screen. The programmer can copy the entire central screen to one of the copy screens or copy only part of the central screen. This allows the programmer to save text for later reference or, the programmers can save part of an on-line manual or a directory listing. Copy screens cannot be edited - they are strictly for reference.

Direct Selection: As the cursor moves from object to object, information about the current object is displayed on the other screen. In one form of direct selection the cursor position is combined with a selection button (on a mouse or on the keyboard). This allows a programmer to move the cursor on to a variable name, select it by pressing the buttons, and receive the data declaration on another screen.

Synchronized Scrolling: This is similar to fusion of the left/right edges. One file is displayed per screen. The scrolling of the files is linked; the command up-one-line moves all files up one line. This is convenient for comparing files with similar lay-out, e.g. versions of same program. Another use is for scrolling the input to a document formatter and the resulting formatted output to check and correct errors. Clever software can make synchronized scrolling even more useful by supporting the scrolling of related items in several files even if the lay-out of the files are not the same, e.g. an input file to test cases and an output file of results.

Independent Scrolling: All screens can be controlled separately. A programmer can look at arbitrary parts of several files or several parts of one file. Independence is similar to logging on to several terminals at once. The advantage is that you can copy from one screen to another. This is the most general, least structured way to use several displays. The screens are just there - available for whatever the user wants to use them for. For the concept to be useful changing screens must be simple and rapid.

These concepts can be applied in many program development situations:

Input - Filter - Output Situations: Using independent screens the input to a program, the program text, and the output of the program can be displayed at once. This makes a very powerful debugging tool. An advanced implementation of synchronized scrolling would allow the programmer to step through the program. As each item from the input is read, the programmer would see exactly which lines of code were executed and what output was produced. Some interpreters currently provide program tracing, but they use overlapping windows. All too often the output in question is obscured by the window listing the program. Programmers must work around these problems by pushing and shuffling windows.

Integration And Reference Tasks: Many tasks require referring to some text or comparing between texts. Help screens are little help when they obliterate the lines you need help with. Multiple screens can make such tasks much easier. For example, a programmer could display on-line manuals and program text at the same time, using independent screens, to look up the correct use of a procedure. Using synchronized scrolling a programmer could display the text of a program on one screen and the documentation on another screen.

Semantically linked parts of different files can be displayed together even if there is no one-to-one correspondence of lines between the files. Thus moving up one line on one file may involve scrolling up several lines on some other file so the linked parts stay together. For example, a programmer could put a program's code in one file and its comments in another. Some method for linking comments to specific parts of code would be provided. Thus, even if comments are larger than the code both will

This gives more room on screen for both code and comments. Also, the code can be read without the distraction of the comments. Finally, the comment file can serve as high-level documentation when read without the code (this requires carefully written comments).

Direct selection is used in reading Pascal programs. When the cursor is placed on a variable (and the button is pressed) the declaration for the variable is displayed. If the cursor is on a procedure name the parameter list (or even the whole procedure) is displayed. Implementations were done by Ruly Arifin, Judd Rogers, and Phil Shafer, with experimental testing conducted by Linda Weldon.

Multi-Screen Editors: New editors can be designed with the extra viewing space of the multi-screen environment in mind. The space can be used to make working with several files at once very easy. This makes the reading, comparing etc. of multiple files much easier as it presents the text in a natural format; not squeezed into an unreadably small window. Such editors make copying parts of one file to another easier since the programmer sees both the source and destination at once.

The editor presents the program text in a cognitively useful manner that does not resemble the format required by the compiler. For example, a procedure, the types it uses and all its calls can be presented to the programmer at once. By opening several screens for a single file and putting parts of the program text in each screen, all this information is presented at once. The compiler enforced format can still be evident to the programmer if line numbers (or some other indication of the actual sequence of the lines) are put in the display by the editor.

Our two screen editor using a standard IBM PC with a color and a monochrome display was implemented by Bob Pollachik. It supports fusion, independent scrolling, copying and synchronized scrolling.

Cooperative Problem Solving: Another use for multiple screens is to have programmers at different sites view the common screens, with communication arranged by a network. They can talk by phone and use cursors to point at text or graphics.

There are several different levels of interaction available. Simple talk programs are common on mainframes. These programs provide telephone like function for people logged on to the computer. However, talk programs are not as productive as a conference telephone call. Short conversations are useful but the medium is inherently too slow for extended conversations (typed text is not as fast or rich as voice for conveying information). More productive interaction is possible if the people can do more than 'talk'. With several screens available, conversants can use one (or part of one screen) to talk and the others to exchange files, send copies of screens, or watch a program run. Gupta Pradeep conducted two experimental evaluations of cooperation modes for program debugging and comprehension tasks. Individual variation in ability dominated the differences, but useful design guidelines emerged from observation of the subjects.

Some Human Factors Issues For Multi-Screen Workstations

Working with a multi-screen terminal introduces a number of new concepts as well as questions about the best way to human factor the system. For example, with several screens how does one refer to a particular screen to direct input to appear there or to copy information from one screen to another? In a command language one could refer to screens in the same way that one refers to other peripherals. For example,

```
copy scra: scrb:
```

would copy the contents of Screen A onto Screen B.

Selection using icons and menus of actions might be powerful but confusing. How do you represent a screen using an icon and where do you put it (which screen)? Nevertheless, some interesting and creative solutions to this problem are no doubt possible. For example, if a pointing device such as a mouse is used, one could display mouse passageways (mouse holes) at the sides of the screen. Dragging the cursor through one hole jumps the cursor to another screen. Dragging the cursor through another hole copies the contents of that screen to another. Other passageways invoke the operations of fusion, automatic selection, and input-filter-output.

In cases were the content of the screens implies differences in operations on the screen, some way of signifying the difference is needed. For example, if one screen is 'live' and another is a copy screen, the user needs visual feedback as to the state of each screen. One possibility is to change the shape of the cursor. Another is to change the background pattern or the shape of a border around the screen. Experiments must be run to decide on the best method.

An obvious human factors problem is the physical separation between the screens. This can be extremely disruptive in cases where one needs to compare the contents of one screen with another, line-by-line. When information is linked, as in fusion, it may be better to use a larger screen rather than multiple screens. In cases where the information is for reference (help screens) or of a different type (input-filter-output), the separation does not cause as much of a problem.

Cognitive Layout in Multi-screen Workstations

The added information that can be displayed on multi-screen workstations can either be beneficial or detrimental depending on the lay out. The surface layout of the information on the screens should match the user's expectation. We use the term cognitive layout' to indicate the user's mental picture of the information. With two screens, the user might see the information as two pages in a book, the left page and the right page. Text editing using fusion and synchronized scrolling is likely to evoke this cognitive layout of a linear array of information.

On the other hand, when one screen is used as a copy screen, a short term memory or notepad layout is appropriate. One screen may be viewed as a working memory, while another is viewed as a temporary store. The memory storage layout is particularly important for systems that transfer files from one application program to another using concepts such as a clipboard or a scrapbook.

Such systems currently have limited screen space and cannot afford to display temporary storage files during other processes. Consequently, users are not always sure just what it is on the clipboard and must trust their own memory. The difficulty is that they have to remember what's there while they are changing mental gears to start a new task. Multi-screen workstations allow constant display of the temporary storage file, relieving the user of the burden of remembering the contents.

A hierarchical or 'blow up' layout is useful when searching for information in a database or keeping track of position in a lengthy manuscript. One screen displays the bird's eye view of the world; the next zooms in on the details. For example, the left screen shows the chapter titles with the current title highlighted. The right screen displays the text of that chapter. The user's cognitive model of the layout is one of progressively finer detail.

Putting the Concepts into Action

We began our work on multi-screen workstations by building a two screen system using the monochrome and color displays on an IBM PC. This system, which supports fusion, synchronized scrolling, copying and independent scrolling, was used for several experimental studies.

Unfortunately, the fonts on the monochrome and color displays are very different. The color display's font is inferior to the monochrome's. Most users find the color display hard to work with. We needed to have all monochrome displays to do further research. While it is possible to put more than one monochrome adaptor in an IBM PC, it is not useful. Since the monochrome adaptors are part of the memory map of the PC and there is no way to make the adaptors move to a different part of the memory map, all monochrome adaptors on the expansion bus of an IBM PC will display the same thing. Each adaptor has four pages of memory that can each hold one screen for display. Unfortunately, the current page displayed is set by writing a byte to a port on the adaptor. Since all adaptors on the bus will respond to writing the byte to the port, there is no way to get one (and only one) adaptor to switch pages. The only way to change this unfortunate situation is to modify the hardware on the monochrome adaptor. We are very reluctant to do this.

Since we could not get two useful monochrome displays on one PC we decided to link two PCs together. This gives us the two monochrome screens displaying the different information that we needed at the cost of having to get the two machines communicating. We decided to use RS-232 ports for the communication as the RS-232 technology is simple to work with and about as fast as we would need.

We made a null modem cable and purchased two RS-232 cards. The null modem cable switches the RS-232 handshaking signals about so that it appears that there is a pair of modems between the two RS-232 ports.

With the hardware connected we proceeded to write the software for the two machines to communicate and appear as one machine to the person using them. A copy of the same program runs on each machine. The copies communicate to keep each other in step.

The programs for our experiments read commands off of the keyboard and execute them. If one machine reads the keyboard and sends the other the command it finds, and both programs then execute that command, neither program can ever get out of step.

We needed a method to send characters from one machine to the other. We designed a simple protocol to exchange single characters across the RS-232 ports. Simple for us meant easy to code and debug. We did not worry about the effect of the protocol on speed of transmission.

The protocol is for the two machines to indicate to each other that they are running by exchanging DTR and DSR signals. The machine that is writing (running the procedure that writes a character) is the data source. The writer sets RTS and waits for the reader to set CTS. Once the writer receives CTS the character is written, sent to the reader by the hardware, and read. The reader stops setting CTS once it successfully reads the character. Both machines now clear DTR and DSR and the exchange is over.

There is a provision for parity checking in the protocol. After the reader reads the character and clears CTS, the parity is checked. If there is an error the reader clears DTR and the procedure exits. If the character is correct the reader sets CTS, waits a bit, clears CTS and then clears DTR and exits. At the same time, after the writer writes the character, it waits for CTS to be cleared by the reader, and then waits again for either DTR to be cleared (an error) or CTS to be set, then cleared, followed by DTR being cleared.

Although we included parity checking in our protocol we have never had a parity error. We have a short length of cable and have the machines set up in a building with no large electrical equipment so there is little chance for parity errors.

All the code involved in moving a character from one machine to another is in three procedures of 'C' code. The three procedures are about three pages long.

Once we had character-at-a-time communication it was rather simple to modify existing programs to work in the new environment. A separate implementation supports cooperative problem solving, using two PCs also linked by the RS-232 ports. We are conducting experimental tests of team program debugging, the importance of a telephone voice link, and different protocols for controlling cursors.

We find that these new environments are appealing to programmers and most users believe that these concepts are beneficial. However, there is clearly a period of accommodation to these novel approaches before the productivity benefits accrue. We have attempted to keep the commands simple and few in number, so as to ease the learning process. During 1986 we will conduct several experiments to ascertain the

effectiveness of these multi-screen concepts for programming tasks and refine the designs.

There are attractive opportunities for expanding the horizon of programmers. Multi-screen workstations offer the potential of showing more relevant information concurrently. Multi-screen systems are relatively cheap to build and offer provocative and novel ways to develop software.

Another effort was the use of the IBM 3290 Plasma Display Workstation. One implementation under XEDIT generated a two column display each having 60 lines of 80 characters. Seeing 120 lines of program text on the screen at once leaves a dramatic impression and changes the way programmers study programs.

A second prototype was for a hierarchical browser which shows the program's modular representation in the upper half of the screen. By pointing at a module, the user produces the code in a scrollable window on the lower half of the screen. This allows the programmer to study the code in a more orderly way, easily jumping through the code to view related modules. We are enthusiastic about the hierarchical browser idea and are planning empirical tests to assess the benefits of several versions.

There is always the danger that more information acts as a distraction and that extra commands can increase confusion. Careful attention must be paid to the user interface design.

Our goal is not to produce software or hardware products, but to develop ideas and validate their effectiveness as we refine our cognitive models of human performance with computers.

2) Menu selection

Menu selection as a mode of user control over the human/computer interface has been the topic of research of a grant supported by Control Data Corporation. This effort has been conducted in collaboration with Prof. Kent Norman of the Department of Psychology. Although menu selection seems like a straight forward method of interaction for novice users, our research indicates that there are a number of important human factors considerations that must be taken into account. A series of empirical studies have been conducted in which menu structures have been implemented using a menu selection prototyping system (MSPS) developed here at Maryland.

The first study investigated training methods in learning a content free menu system. Subjects were required to find target items in a menu tree with 3 choices at each of three levels. The labels of the alternatives at the top and middle levels were meaningless words (similar to many real world menu systems as perceived by the novice user). The results indicated that users who studied the global tree of the menu do somewhat better than users who (a) memorize sequences of choices, (b) study menu frames, or (c) traverse the menu in a trial and error fashion.

The second study investigated training methods and distinctiveness in searching for target items in a commercial timesharing service. The entire menu structure was implemented on the MSPS in two forms. One form was the original system. The other form changed a number of the alternatives so that they would be more distinctive. Again it was found that the global tree method of training led to the best performance. Distinctiveness had a marginal effect. In cases where the user searched for a specific target item, times were faster than when searching for a item to meet a requirement or a function. In the latter case, additional time is required to determine if the found item meets the needs of the user.

A third study investigated the effect of menu structure on the search process. Previous work indicated that breadth is superior to depth of the tree. Depth tends to bury items and lengthen the path of selections to get to an item. However, if the depth of the menu is held constant does it matter if the breadth varies along the way? A data base of 256 gift items was generated and clustered into 5 different structures with four levels. The number of choices at each level in the 5 structures were as follows: 4-4-4-4, 2-2-8-8, 8-8-2-2, 2-8-8-2, 8-2-2-8. Results to date indicate a superiority for the 4-4-4-4 and 8-2-2-8 structuress over the others in terms of overall time to find items and number of frames visited.

Other work along these lines has involved the development of a menu evaluation scale in which users assess the degree to which the menu facilitates or retards work by its clarity, speed, efficieny of path, etc. In addition a theory of user search behavior is being developed that relies on behavior choice theory and information theory. It is hoped that this theory in conjunction with user assessments of the utility and meaningfulness of items will have predictive power in terms of evaluating user performance over time.

Future work is planned along three lines. First, it is important to understand the efficient use of menus by experienced users. Studies are being planned in which users search behavior will be monitored over extended periods of time. Second, the transfer of training from one system to another is becoming important as users are having to learn a number of different systems. Another study is planned in which users will learn one system and then transfer to another having limited compatibility. Finally, more and more systems are using icons in addition to text. Since icons are highly recognizable and discriminable they may greatly facilitate the use of menu systems in which the user does not need to recall the item (icon) but merely recognize it in a display of many icons. Studies are planned in which icons will be assessed in terms of recognizability and connotation. An experiment will then be designed in which we compare verbal, iconic, and iconic plus verbal menus.

3) TIES and OLMM

The Interactive Encyclopedia System (TIES) has been under development at the University of Maryland since Fall 1983. It allows novice users to explore information resources in an easy and appealing manner. They merely touch (or use arrow keys to move a light bar onto) topics that interest them and a brief definition appears at the bottom of the screen. The users may continue reading or ask for details about the selected topic. An article about a topic may be one or more screens long. As users traverse articles, TIES keeps the path and allows easy reversal, building confidence and a sense of control. Advanced features include the ability to view an index of articles or print out articles of interest.

TIES is attractive for instruction (and entertainment) because the author's ideas and writing style are the focus of attention. Through careful human factors design, the computer aspects have been trimmed to let the author communicate to the students and to allow the students to control their learning.

The current version of TIES is being produced at the University of Maryland for the U.S. Holocaust Memorial Museum and Education Center under contract from the Department of Interior. This version will include:

- Novice user browsing software
- Database of approximately 110 articles (100-500 words each) on "Austria and the Holocaust 1933-1945", written by Dr. Marsha Rozenbilt of the History Department.
- Authoring software for composing new articles and editing

TIES is appealing to authors because of the explicit instructional model, the reduction of computer-related concepts, the focus on content, and the lively user interface. It is an engaging challenge to reformulate pedagogic plans into the network of related articles that TIES supports. There is a great sense of satisfaction in composing articles and seeing the linkages come to life as they are used by students in novel ways.

TIES allows authors to create a network of conceptual knowledge in which concepts are linked associatively and the learner is free to explore pathways based on their needs and interests. Potential guides for any discipline, travel guidebooks, annotated Shakespeare or the Bible, and maintenance manuals for equipment. Each visitor suggests intriguing and novel applications.

TIES was implemented by Dan Ostroff under the direction of Ben Shneiderman and Dr. Janis Morariu of the College of Library and Information Services. It runs on a standard IBM PC (monochrome or color) and on IBM PCs equiped with touchscreens. We are attracted to the possibility of eliminating the keyboard while still providing substantial exploratory power. TIES was first written in APL and has been rewritten in the C programming language. A brief user's guide and a more extensive author's manual are available in paper form and as TIES databases.

Three experimental studies have been conducted to test out certain design alternatives (such as demonstrating the advantage of arrow keys over the mouse for this system) and observe user behavior. More than 160 subjects participated in these controlled experiments. In addition, more than two hundred novices and experts have tried and commented informally on the current design.

In the study comparing the arrow keys (maybe better termed "jump" keys because the cursor would jump to the closest target in the direction pressed) to the mouse, the arrow keys proved to be and average of 15% faster and preferred by almost 90% of the subjects. We conjecture that when there are a small number of targets on the screen and when jump keys can be implemented, they provide a rapid, predictable, and appealing mechanism for selection.

In a second study using the TIES technique, subjects traversed a database with 42 articles about the University of Maryland Student Union. The embedded menus technique reduced the number of screens viewed when compared with an explicit menu strategy. There were significant reductions in the times for task performance, and the subjective preference was strongly for the embedded menus.

The embedded menus idea was also used for two experiments with online maintenance manuals (OLMM), conducted by Larry Koved and supported by IBM Federal Systems Division. A tree structured and linear form of a 52 page maintenance manual was prepared for screen presentation and in paper form. Experimental subjects had to perform 12 tasks using one of the manuals. Significant differences were found showing that time was reduced using the paper versions. No significant differences were found between the tree and linear versions for speed or error rates. When a pruning algorithm was applied to the text to allow users to trim text unrelated to their task, the time was cut in half. This latter experiment used only the computer condition and demonstrated one of the advantages of screens over printed text. This is important, since for many applications printed manuals are still easier to use and approximately 30% faster to read than computer displays.

A field trial was conducted for a week in the B'nai Brith Klutznick Museum in Washington, DC using the touchscreen and arrow key versions. Visitors from 7 to 77 years explored the Austria database and provided comments on subjective evaluation forms. Reaction was generally very positive with a strong preference for the touchscreen version.

TIES is complete, but there are small refinements and many potential extensions which we would like to pursue. The software needs further documentation and the authoring guide could be expanded.

Major extensions include support for graphics, videodisc, or integration with other software. Further testing is needed to select the optimum touchscreen or other input devices and to test alternate screens. We are also interested in further testing with the screen mounted horizontally inside a nicely built wooden table. We hope to attract the large fraction of the population that is anxious about using computers, but could benefit from the information resources of a computer system.

4) Direct manipulation and DMDOS

Certain computer systems generate feelings of enthusiasm, confidence, desire for exploration, clarity, competence, and predictability. These positive experiences seem to emerge when the user is presented with a visual display of the world of action with the objects of interest clearly available for intuitively obvious manipulation. Operations are accomplished by physical actions, such as special buttons, joystick, mouse, or touchscreen, rather than by typing commands or making menu selections. These operations are generally rapid, incremental (that is, smooth or continuous), and reversible.

Familiar examples of direct manipulation systems include full screen display editor-formatters (What you see is what you get - WYSIWYG), video games, VisiCalc and its descendants, some educational games, air traffic control displays, and the Macintosh, Lisa, and XEROX STAR environments. TIES might also

be seen as a direct manipulation system for pursuing ideas in textual databases.

To explore the design issues in direct manipulation systems Osamu Iseki (Visiting Scholar for Nippon Electric Company in Japan) implemented a direct manipulation version of the IBM PC-DOS commands.

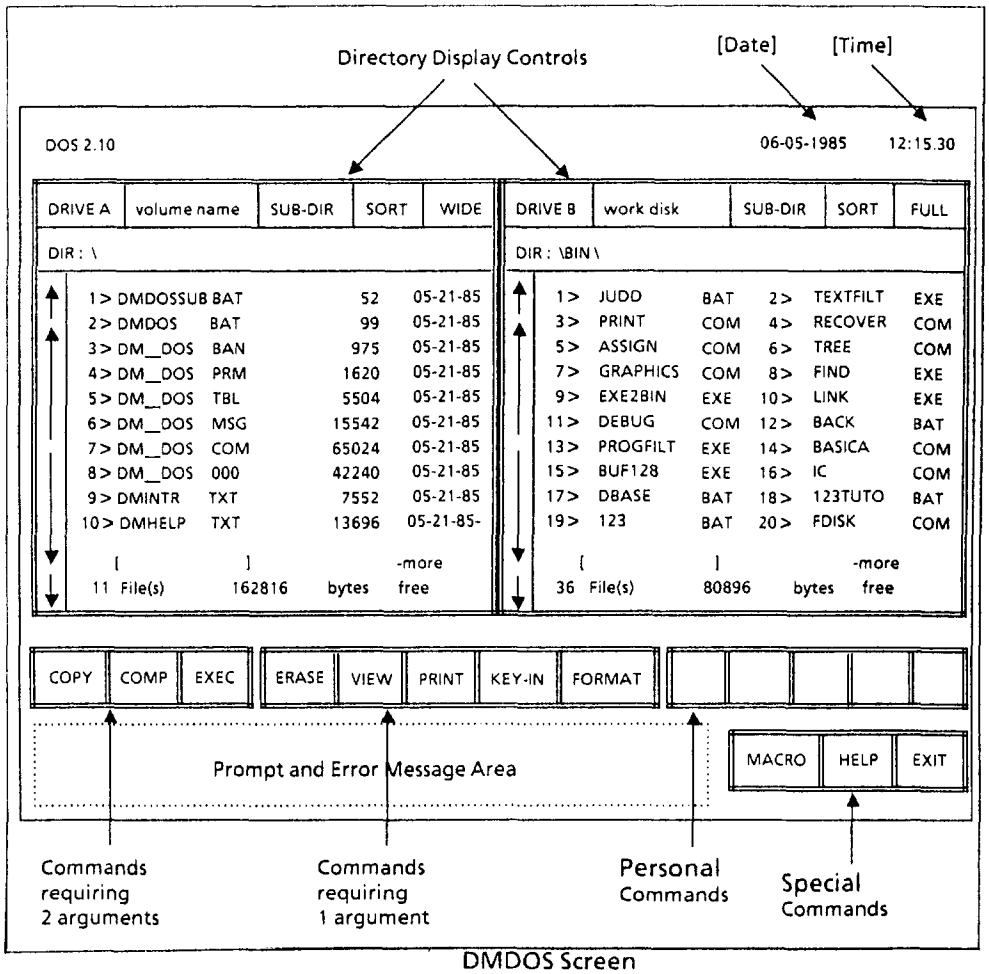
Called DMDOS (for Direct Manipulation DOS), it displays both the A and B directories simultaneously. Directory files are selected by pointing and clicking. The directory can be sorted, sub-directories can be traversed, and the display can be switched from WIDE to FULL mode.

Operations supported are comparison of two files, copying to a file, copying from a file to the screen, copying from the screen to a file, copying from a file to the printer, comparison of two files, erasing of files, execution of programs, and online help.

To copy a file from one disk directory to another, merely point (using arrow keys or the mouse) at the file, and click (press RETURN or the mouse button). Then point and click on the COPY button. Finally, point at the free space on the second directory and click once. Now, you may type the new file name or just click a second time to use the same name. Your file is copied.

Design refinements have been based on usage experience, comments from dozens of knowledgeable users, and the reactions of 24 subjects in an experimental comparison. In this experiment non-programmers were taught either PC-DOS or DMDOS and required to carried out a benchmark set of tasks. The subjects were struggling to absorb the concepts in DOS, such as files, disks, directories, copying, etc., and DMDOS users were faster but not at a statistically significant level.

We are continuing to refine and test DMDOS to understand the relative merits of specific changes. A macro facility which allows creation of batch files by merely carrying out operations is being added. An empirical test with the mouse and with more knowledgeable users is being considered.



DMDOS Screen