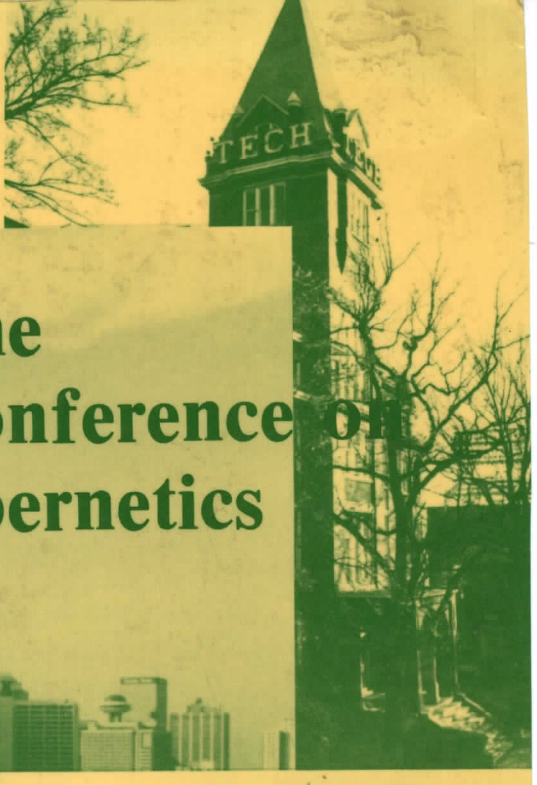
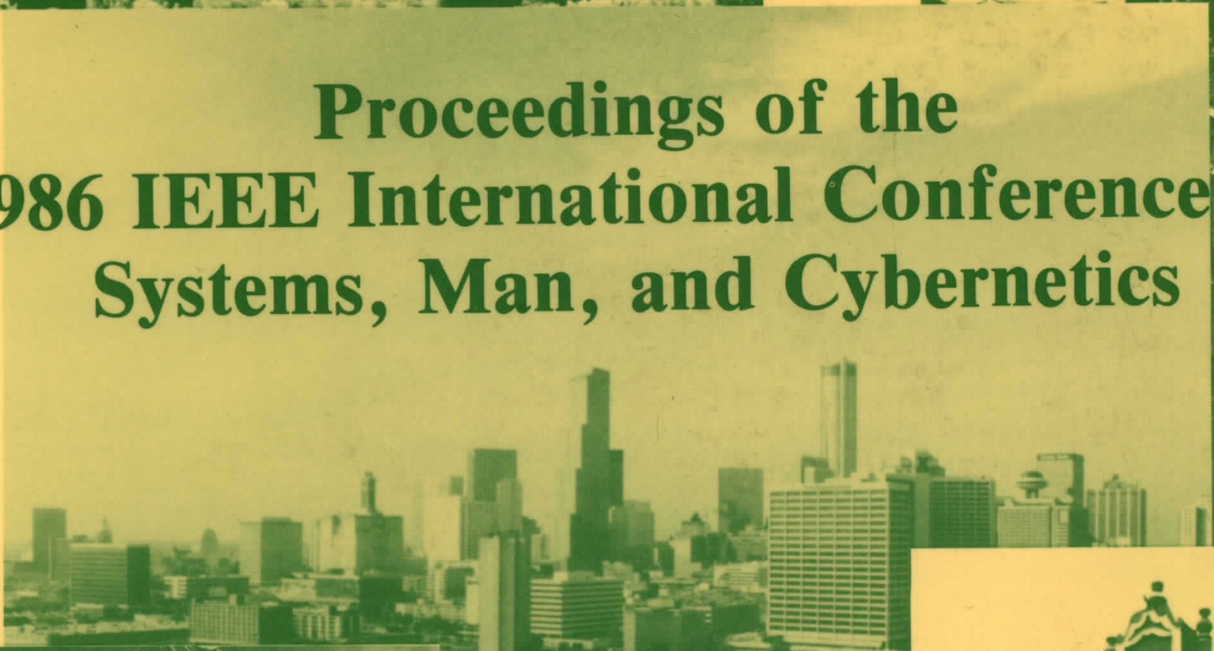
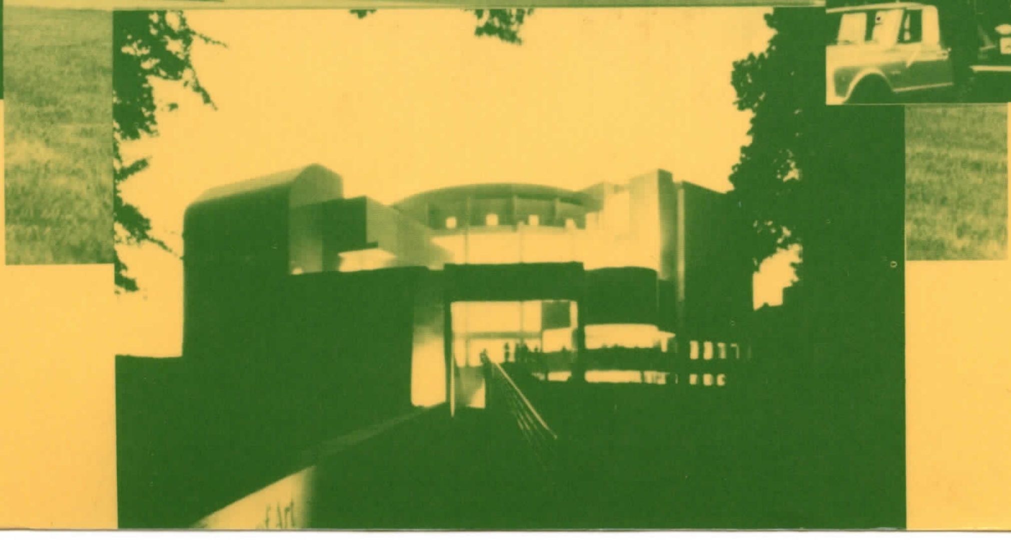


Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics



Atlanta, Georgia
October 14-17



Vol 1

86CH2364-8

Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics

Pierremont Plaza Hotel and Conference Center
Atlanta, Georgia
October 14-17, 1986

86CH2364-8

Additional copies may be obtained from:
IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854

IEEE Catalog No. 86CH2364-8

Library of Congress Catalog No. 86-81599

Direct Manipulation

Robert J.K. Jacob

Naval Research Laboratory
Washington, D.C. 20375

Abstract. Direct manipulation is a powerful new paradigm for designing user interfaces. At the core, this approach presents visual representations of a set of objects on a screen and provides a user with a standard repertoire of manipulations that can be performed on any of them. This means that the user has no command language to remember beyond the standard set of manipulations, few noticeable changes of mode (i.e., most commands can be invoked aι any time), and a continuous reminder of the available objects and their states on the display. This paper examines the nature of direct manipulation user interfaces and some issues associated with them. It distinguishes interfaces that operate directly on concrete visual objects from those that use visual metaphors to operate in more abstract domains. Two examples of direct manipulation interfaces are described, and some benefits and drawbacks characteristic of direct manipulation are considered. It is concluded that, while the prospects for combining direct manipulation with "intelligent" user interfaces are promising, little research has been conducted in this area to date.

Introduction

The concept of Direct Manipulation user interfaces, introduced by Ben Shneiderman,[17] brought together a collection of principles and techniques shared by a number of innovative user interfaces that were widely regarded as easy—and enjoyable—to learn and use. Shneiderman identified the defining characteristics of direct manipulation user interfaces:[17]

- Continuous representation of the object of interest.
- Physical actions (movement and selection by mouse, joystick, touch screen, etc.) or labeled button presses instead of complex syntax.
- Rapid, incremental, reversible operations whose impact on the object of interest is immediately visible.
- Layered or spiral approach to learning that permits usage with minimal knowledge.

The essence of such a user interface is that the user seems to operate directly *on* the objects in the computer rather than carrying on a dialogue *about* them. Instead of using a command language to describe operations on objects, the user "manipulates" objects visible on a display. For example, to delete a file named *FOO* in a command language system, the user would ask the computer to *delete the file whose* name *is FOO.* In a direct manipulation system, he or she would find a representation of the file on the screen and "delete" it directly (perhaps with a delete button, a mouse gesture, or the like). The effect would be apparent immediately on the display. Further, the user could apply the same delete operation to any other object visible on the display (provided the system permits its deletion); the user does not have to learn new commands for them.

The most visible characteristic of a direct manipulation user interface is this ability to manipulate displayed objects. Hutchins, Hollan, and Norman[8] identify that quality as *direct engagement.* A direct manipulation user interface typically provides a set of objects presented on a display and a standard repertoire of manipulations that can be performed on them. There is no command language for the user to remember beyond the set of manipulations, and generally any of them can be applied to any visible object. The displayed objects are active in the sense that they are affected by each command issued; they are not the fixed outputs of one execution of a command, frozen in time. They are also usable as inputs to subsequent commands.

However, the ultimate success of a direct manipulation interface requires more than this manipulability. It also requires directness in the form of low *cognitive distance*,[8] the mental effort needed to translate from the input actions and output representations to the operations and objects in the problem domain itself. The visual images chosen to depict the objects of the problem or application domain should be easy for the user to translate to and from that domain. Conversely, the actions required to effect a command should be closely related to the meaning of the command in the problem domain. The effective use of direct manipulation hinges on the choice of a good metaphor for representing the world of the application in terms of screen images and input actions.

Categories of Direct Manipulation Interfaces

The problem domains with which direct manipulation user interfaces must deal can be divided into two classes. In the first, the underlying problem is concerned with static, concrete objects; examples include menus, screen layouts, printed forms, engineering drawings, typeset reports, and fonts of type. A direct manipulation user interface for operating in such a domain can simply use a picture of the concrete object in a "what you see is what you get" style of editor. The choice of representation is straightforward, since there is already an agreed-upon concrete visual form for the objects of the problem area. Unfortunately, this approach is only possible where there can be a one-to-one correspondence between the problem domain and the visual representation.

A more difficult problem arises in the second class of direct manipulation user interface. Here, the domain involves abstract objects, which do not have a direct graphical image, such as time sequence, hierarchy, conditional statements, frame-based knowledge, or data in a data base. To provide a direct manipulation interface for such a domain, it is necessary first to devise a suitable graphical representation or visual metaphor for the objects. "What you see is what you get" is still helpful, but not sufficient to solve this problem, since the objects are abstract.

Interfaces for Concrete Objects

Examples of direct manipulation user interfaces for problem domains in the first class, where a concrete representation is available, often involve systems for creating and manipulating graphical images. A screen editor is a good example.[19] A more elaborate example can be found in a typical direct manipulation editor for a typesetting system, which displays a mockup of the final printed page on a high-resolution display.[9] It permits the user to manipulate the displayed page and, as he or she does, it immediately redisplays the portions of the page affected by the change. Since the system is designed for producing printed pages, the visual representation chosen for the direct manipulation interface is simply a high-resolution picture of such a page. Other examples of this class of direct manipulation user interface include:

- Font editors, where the visual representation is a picture of the character being edited.
- Computer-aided design or drafting systems based on standard engineering drawings, where the problem domain is that of paper drawings and the visual representation is a screen image of (portions of) the same drawings.
- Systems for designing printed forms, where, again, the visual representation is simply a screen picture of the form.

The objects represented in these direct manipulation interfaces are concrete physical objects, and most of the manipulations available to the user are based on common physically-understandable actions such as moving, copying, changing size, or removing. These systems therefore generally have small cognitive distances between the problem domain and the objects and operations of the user interface.

Interfaces for Abstract Domains

More difficult problems arise where the problem domain is abstract. It is helpful if a reasonable concrete representation for the abstract problem domain is already in use, perhaps in pencil-and-paper form, and it can be exploited by the user interface designer. For example:

- A system for manipulating a geographic data base might use choropleth ("patch") maps as its visual representation for the data and allow the user to manipulate the maps to view the data.[1]
- A computer-aided manufacturing system that drives machine tools directly can use conventional mechanical drawings as its visual representation.
- A matrix or spreadsheet calculator. The visual metaphor widely chosen for such systems is the accountant's paper worksheet, with its rows and columns of figures. It is a concrete visual representation of an abstract domain (matrices). The rapid acceptance of direct manipulation spreadsheet systems shows that the paper spreadsheet was a particularly fortuitous choice of visual metaphor for this abstract object.
- STEAMER[7] allows its user to operate a simulated steam engine. The problem domain is the sequence of operations for running a steam engine, and the visual representation is an image of an engine control panel, with some extensions not available on conventional panels.
- The Xerox Star desktop manager[18] and its numerous philosophical descendants, such as Apple Macintosh, handle a domain of computer files and directories. For their visual metaphor they use a stylized picture of a desk surface with papers, file folders, trays, and even a waste basket to represent the computer file domain. A familiar operation, like moving a paper from a file folder to the waste basket, corresponds in an intuitively plausible way to deleting an item from a data file.
- Finally the system for designing user interfaces described later has as its problem domain the time sequence or syntax of input and output operations performed by a system and its user. The visual metaphor chosen here is the state transition diagram, already used to represent user interface designs on paper. The designer manipulates a picture of a state transition diagram to describe and effect changes in the behavior of the user interface being constructed.

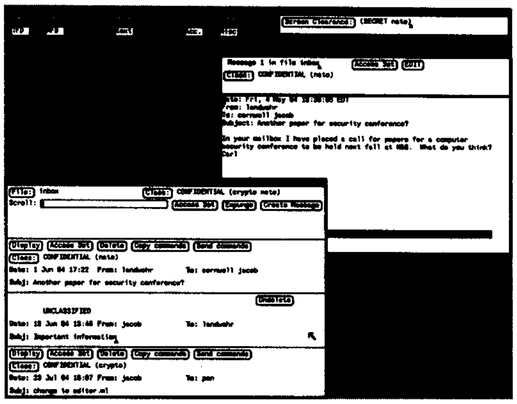
In each of these systems, an abstract problem domain was represented by a concrete visual metaphor. The visual metaphors used in these examples were all existing concrete objects, rather than newly-invented representations. Each representation was chosen to minimize cognitive distance between problem and representation, so that when the user manipulates the objects in the representation, his operations are closely allied to those of the problem domain. The most successful direct manipulation user interfaces thus far have depended on a fortuitous or perspicacious choice of visual metaphor.

It is also possible to design a direct manipulation interface by inventing a new visual object and teaching it to users as the representation of some abstract domain. Inventing good representations is a difficult problem, and there are relatively few examples of this class of direct manipulation system:

- The Spatial Data Management System[6] uses an hierarchical collection of icons arrayed in a map-like layout to depict data in a data base. The user examines the data base by panning across the layout or zooming up or down in the hierarchy. The visual metaphor for the data base data is thus a new pictorial representation. The original icons in the system were individually sketched; methods for generating graphics directly from the data were also studied.[3]
- DMDOS is a user interface to the IBM PC-DOS operating system command language developed by Ben Shneiderman and Osamu Iseki at the University of Maryland.[10] It uses a tabular representation of directories, disks, peripherals, and other objects of interest and permits the user to operate directly on the items shown in the display.

A Direct Manipulation Military Message System

Figures 1 and 2 show the display of a direct manipulation military message system, one of a family of prototype message systems being built in the Secure Military Message Systems Project at the Naval Research Laboratory.[5] Such a message system is much like a conventional electronic mail system, except that each message (actually, each field of each message), each file, and each user terminal has a security classification. The basic visual metaphor chosen for the direct manipulation version of the message system is simple: a paper message. A message is represented by a screen image that is similar to a traditional paper military message. A new object, a file of messages, is also introduced. This is represented by a display of a list of the summaries (called *citations*) of the messages in the file. Some elements of each message citation in such a display can be changed directly by typing over them; they are indicated by borders around their labels. Other elements are fixed because the application requires it (e.g., the user cannot change the date of a message that has already been sent). In addition, each citation contains some *screen buttons*, or small, labeled boxes. Pressing a mouse button while pointing to such a box causes the action indicated in the box to occur. All the commands that the user could apply to a given message are shown on its citation as buttons. If the user sees a message on the screen, he does not need to refer to a manual to find out what commands he could apply to it.



Date: Fri, 4 May 84 [illegible] EDT
From: landwehr
To: cornwell jacob
Subject: Another paper for security conference?

In your mailbox I have placed a call for papers for a computer security conference to be held next fall at NBS. What do you think?
Carl

Class: CONFIDENTIAL (nato)
Date: 1 Jun 84 17:22 From: landwehr To: cornwell jacob
Subj: Another paper for security conference?

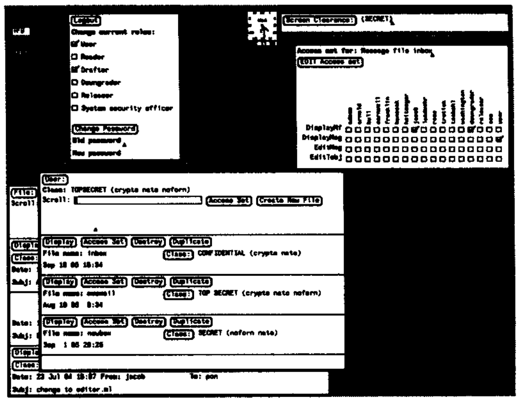
UNCLASSIFIED
Date: 13 Jun 84 13:46 From: jacob To: landwehr
Subj: Important information

Class: CONFIDENTIAL (crypto)
Date: 23 Jul 84 16:07 From: jacob To: pan
Subj: change to editor.ml

Figure 1. Direct manipulation military message system prototype, showing message window (top) and message file window (bottom). *(Security classifications shown are simulated for demonstration only).*

A Direct Manipulation System for Designing User Interfaces

The next example involves an abstract domain: the design of user interfaces (not necessarily direct manipulation ones). That is, the design of the sequence of inputs and outputs, or syntax, of a user-computer dialogue. The visual representation chosen for this abstract domain is one already used for representing user interfaces outside the realm of direct manipulation—state transition diagrams.[11,16] It is thus an appropriate choice for use in a direct manipulation user interface as the concrete visual representation for the abstract notion of syntax. The specific language chosen for use here has been used to specify and directly implement user



Class: TOPSECRET (crypto nato noforn)

File name: inbox Class: CONFIDENTIAL (crypto nato)
Sep 18 85 18:34

File name: [illegible] Class: TOP SECRET (crypto nato noforn)
Aug 10 85 8:34

File name: newbox Class: SECRET (noforn nato)
Sep 1 85 20:26

Date: 23 Jul 84 16:07 From: jacob To: pan
Subj: change to editor.ml

Figure 2. Direct manipulation military message system prototype, showing user roles, access set editor, and message file directory windows.

interfaces for several prototype systems;[13,14] similar languages have also been used by others for this purpose.[2,20] The language used here is part of a methodology for designing and specifying user interfaces;[12,13] the direct manipulation version of it described below is currently being implemented.[15]

An interactively editable picture of this state transition diagram is used as the visual representation of the user interface syntax. The programmer enters the state diagrams with a graphical editor and affixes the necessary labels and actions. As the diagrams are edited, they can be directly executed. Thus the direct manipulation interface provides two types of windows. One shows a demonstration of the sequence being programmed, while the other allows the user to manipulate the visual representation of its syntax. As shown in Figure 3, the simulator window on the left shows the newly-designed user interface (of a simple line-oriented desk calculator program) as it would appear to the user and allows the programmer to interact with it in the role of its user. The programming windows on the right show the state diagrams themselves and allow the programmer to modify them.

Characteristics of Direct Manipulation Interfaces

The principal advantages of direct manipulation user interfaces are psychological: they decrease the demands on the user's short- and long-term memory. For long-term memory, they require remembering only a few generic commands in the form of general-purpose manipulations, which can be applied to most visible objects. Once the users memorizes this set, most specific operations can be derived from it, in contrast to traditional systems with many specific commands to remember. Short-term memory load is reduced because most commands that change the values of objects are reflected immediately in changes in the display of those objects. The system thus continuously displays much of its internal data, rather than requiring the user to remember them and ask specific questions about them.

Direct manipulation user interfaces also introduce some difficulties. They are, both for concrete and abstract objects, more rigidly fixed at a single level of abstraction than command language systems. For example, consider a desktop

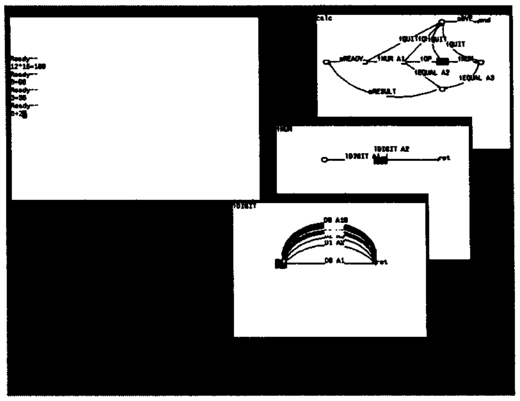


Figure 3. Direct manipulation system for designing user interfaces, showing state transition diagrams for the user interface being designed (at right) and the new user interface itself (at left).

manager like that of the Xerox Star.[18] It deals with an underlying file system at a particular—though generally appropriate—level of abstraction. Low-level details, such as allocating space for new files, reclaiming free space, determining disk layout, and reading and writing directories, are handled conveniently by the system and are invisible to its user. The user operates at the level of whole files and directories, not their component bits, tracks, or sectors. It is, however, difficult for the user to move further up smoothly in abstraction within the same language—perhaps to operate on aggregates of files from different directories or to operate on commands about files. For example, the system makes it easy to move file *foo* to folder *bar* without worrying about lower-level details. But it would be difficult to move all files whose contents contain the string "Star" from one folder to another. The problem is that, while direct manipulation interfaces abstract away a host of irrelevant details up to a particular level, they remain stuck at that level; it is difficult to move up any further. This makes them convenient for users who happen to want to operate at the level provided but cumbersome at either higher or lower levels. By contrast, command language user interfaces encourage such abstraction because they provide a natural means to express it. Many command languages have good, almost intrinsic facilities for abstraction, control structures, formal parameters, combining individual commands, and the like. If a direct manipulation interface is designed for the level of abstraction its user wants to use, he or she will indeed find it convenient. For both direct manipulation and command language user interfaces, that level can be as high or low as desired. The problem arises when the user wants to change levels smoothly. Command languages make it easier to move up or down in level; they encourage abstraction. Direct manipulation interfaces are often stuck at one level of abstraction, be it high or low.

Another drawback applies to the user interface programmer rather than the user. While direct manipulation can make a system easy to learn and to use, such a user interface is generally difficult to construct. Existing examples have required considerable highly machine-specific, low-level programming. Higher-level abstractions for dealing with this new interaction technique are needed.

Modes in the User Interface

Modes or states refer to the varying interpretation of a user's input. In each different mode, a user interface may give different meanings to the same input operations. Some use of modes is necessary in most user interfaces, since there are generally not enough distinct brief input operations (e.g., single keystrokes) to map into all the commands of a system. A moded user interface requires that the user remember (or system remind him) of which mode it is in at any time, and he or she must remember the different commands or syntax rules applicable to each mode. Modeless systems do not require this; the system is always in the same mode, and inputs always have the same interpretation.

Direct manipulation user interfaces appear to be modeless. Many objects are visible on the screen; and at any time the user can apply any of a standard set of commands to any object. The system is thus always in the same "universal" or "top-level" mode. This is approximately true of some screen editors, but for most other direct manipulation systems, where the visual representation contains more than one type of component, this is a misleading view. It ignores the input operation of moving the cursor to the object of interest. A clearer view suggests that such a system has many distinct modes. Moving the cursor to point to a different object is the command to cause a mode change, because once it is moved, the range of acceptable inputs is reduced and the meaning of each of those inputs is determined. This is precisely the definition of a mode change. For example, moving the cursor to the **Display** screen button in the message system example should be viewed as putting the system into a mode where the meaning of the next mouse button click is determined (it displays that message) and the set of permissible inputs is circumscribed (for example, keyboard input could be illegal or ignored). Moving the cursor somewhere else would change that mode.

If direct manipulation user interfaces are not thus really modeless, why do they appear to have the psychological advantages over moded interfaces that are usually ascribed to modeless ones? The reason is that they make the mode so apparent and so easy to change that it ceases to be a stumbling block. The mode is always clearly visible (as the location of a cursor), and it has an obvious representation (simply the echo of the same cursor location just used to enter the mode change command), in contrast to some special flag or prompt. Thus, the input mode is always visible to the user. The direct manipulation approach makes the output display (cursor location to indicate mode) and the related input command (move cursor to change mode) operate through the same visual representation (cursor location). At all times, the user knows exactly how to change modes; he or she can never get stuck. It appears, then, that direct manipulation user interfaces are highly moded, but they are much easier to use than traditional moded interfaces because of the direct way in which the modes are displayed and manipulated.

The Intelligent User Interface

An "intelligent" user interface should be able to describe and reason about what its user knows and conduct a dialogue with the long-term flow and other desirable properties of dialogues between people. It maintains and uses information about the user and his or her current state of attention and knowledge, the task being performed, and the tools available to perform it.[4] It can use this information to help interpret a user's inputs and permit them to be imprecise, vague, slightly incorrect (e.g., typographical errors) or elliptical. It can control the presentation of output based on its model of what the user already knows and is seeking and remove information irrelevant to his current focus. Knowledge-based techniques can also be applied to improve the selection, construction, and layout of graphical outputs.

Most research to date on the processes needed to conduct such "intelligent" dialogues has been applied to natural language, but it is important to realize that such techniques of the intelligent user interface are by no means restricted to natural language. There is no reason why such ideas could not be used in a direct manipulation dialogue. The user's side of such a dialogue can consist almost entirely of pointing and pressing mouse buttons, and the computer's, of animated pictorial analogues. Nevertheless, a dialogue in such a language could exhibit the intelligent user interface properties cited—following focus, inferring goals, correcting misconceptions. This combination of such intelligent user interface techniques with a direct manipulation type of language holds promise for providing a highly effective form of man-machine communication.

Conclusions

This paper has examined direct manipulation interfaces and divided them into two classes: those that operate in a domain of concrete visual objects and those that use such visual objects as proxies to operate in a more abstract problem domain. For interfaces in the second class, the choice of a good visual representation is critical. Some advantages and disadvantages of direct manipulation were also considered. It is noteworthy that the advantages were based on fundamental characteristics of the users, while the disadvantages were largely technical and likely to yield to further research. Finally, the future possibilities offered by combining "intelligence" with direct manipulation were seen to be promising.

Acknowledgments

I want to thank Ben Shneiderman for introducing the idea of direct manipulation and helping me understand it and Don Norman for his suggestions and comments on this work, which have helped to clarify the idea.

This work was supported by the Space and Naval Warfare Systems Command under the direction of H.O. Lubbes.

References

1. J. Dalton, J. Billingsley, J. Quann, and P. Bracken, "Interactive Color Map Displays of Domestic Information," *Computer Graphics* **13**(2) pp. 226-233, Chicago (1979).
2. M.B. Feldman and G.T. Rogers, "Toward the Design and Development of Style-independent Interactive Systems," *Proc. ACM SIGCHI Human Factors in Computer Systems Conference* pp. 111-116 (1982).
3. M. Friedell, "Automatic Synthesis of Graphical Object Descriptions," *Computer Graphics* **18**(3) pp. 53-62, Minneapolis (1984).
4. P. Hayes, E. Ball, and R. Reddy, "Breaking the Man-Machine Communication Barrier," *IEEE Computer* **14**(3) pp. 19-30 (1981).
5. C.L. Heitmeyer, C.E. Landwehr, and M.R. Cornwell, "The Use of Quick Prototypes in the Military Message Systems Project," *ACM SIGSOFT Software Engineering Notes* **7** pp. 85-87 (1982).
6. C.F. Herot, R. Carling, M. Friedell, and D. Kramlich, "A Prototype Spatial Data Management System," *Computer Graphics* **14**(3) pp. 63-70 (1980).
7. J.D. Hollan, E.L. Hutchins, and L. Weitzman, "STEAMER: An Interactive Inspectable Simulation-Based Training System," *The AI Magazine* **5**(2) pp. 15-27 (1984).
8. E.L. Hutchins, J.D. Hollan, and D.A. Norman, "Direct Manipulation Interfaces," in *User Centered System Design: New Perspectives in Human-computer Interaction*, ed. D.A. Norman and S.W. Draper, Lawrence Erlbaum, Hillsdale, N.J. (1986).
9. Interleaf, Inc., "Workstation Publishing Software," , Cambridge, Mass. (1984).
10. O. Iseki and B. Shneiderman, "Applying Direct Manipulation Concepts: Direct Manipulation Disk Operating System (DMDOS)," *ACM SIGSOFT Software Engineering Notes* **11** pp. 22-26 (1986).
11. R.J.K. Jacob, "Using Formal Specifications in the Design of a Human-Computer Interface," *Comm. ACM* **26** pp. 259-264 (1983).
12. R.J.K. Jacob, "Executable Specifications for a Human-Computer Interface," *Proc. ACM SIGCHI Human Factors in Computer Systems Conference* pp. 28-34 (1983).
13. R.J.K. Jacob, "An Executable Specification Technique for Describing Human-Computer Interaction," pp. 211-242 in *Advances in Human-Computer Interaction*, ed. H.R. Hartson, Ablex Publishing Co., Norwood, N.J. (1985).
14. R.J.K. Jacob, "Designing a Human-Computer Interface with Software Specification Techniques," pp. 139-156 in *Empirical Foundations of Information and Software Science*, ed. J.C. Agrawal and P. Zunde, Plenum Press, New York (1985).
15. R.J.K. Jacob, "A State Transition Diagram Language for Visual Programming," *IEEE Computer* **18**(8) pp. 51-59 (1985).
16. D.L. Parnas, "On the Use of Transition Diagrams in the Design of a User Interface for an Interactive Computer System," *Proc. 24th National ACM Conference* pp. 379-385 (1969).
17. B. Shneiderman, "Direct Manipulation: A Step Beyond Programming Languages," *IEEE Computer* **16**(8) pp. 57-69 (1983).
18. D.C. Smith and others, "Designing the Star User Interface," *Byte* **7**(4) pp. 242-282 (1982).
19. R.M. Stallman, "EMACS: The Extensible, Customizable, Self-documenting Display Editor," MIT Artificial Intelligence Laboratory, Cambridge, Mass. (1979).
20. A.I. Wasserman and D.T. Shewmake, "The Role of Prototypes in the User Software Engineering (USE) Methodology," pp. 191-209 in *Advances in Human-Computer Interaction*, ed. H.R. Hartson, Ablex Publishing Co., Norwood, N.J. (1985).