

Published as:

Witten, I. H. and Greenberg, S. (1993). User Interfaces. In A. Ralston and E. D. Reilly (Eds.), *Encyclopaedia of Computer Science*, pp. 1411-1414. Van Nostrand Reinhold, New York.

USER INTERFACES

Ian H. Witten and Saul Greenberg

The *user interface* is that part of a computer system through which human user and computer communicate. With the increasing prevalence of interactive personal computer systems the importance of human communication is growing steadily, and many systems now stand or fall on the quality of their interfaces. Interfaces consume a large amount of software construction and maintenance effort—estimates of the fraction of an interactive system's code devoted to the user interface vary from one third to almost two thirds.

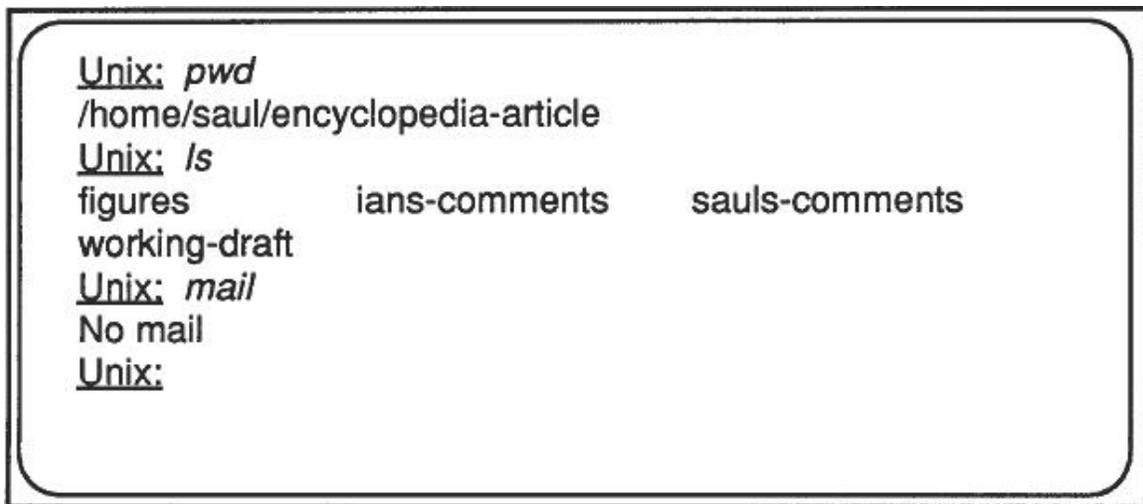
User interfaces evolved from the *job control languages* available on batch computing systems that allowed users to describe to the system the requirements of their tasks. These provided facilities for users to identify themselves to the system for security and accounting purposes; inform the computer about the resources required by tasks; specify input/output devices and files needed; and determine what action the computer should take in the event of error. Interactive systems renamed their job control languages *command languages*. These environments simplified some aspects of the human/computer interface. A stream of commands could be submitted and interpreted one line at a time, which allowed users to respond to evolving situations rather than forcing them to anticipate all conditions. The computer could to some extent take the initiative in the dialogue, prompting the user for whatever information it needed. In other respects interactive environments complicated the interface, for a great many new facilities became possible and were absorbed into the command language (examples include file management, interactive editing, social functions like getting information about users and processing electronic mail).

User interfaces changed dramatically when interactive systems were liberated from the tyranny of the teletype—although change came slowly and, at first, uncertainly. The advent of cursor-addressable VDUs allowed the temporal sequence of events, imposed on the user by a command language, to be relaxed in favour of business-form-style spatial layouts that gave users control over the sequence in which items were entered. The widespread use of bitmapped graphic displays provided opportunities to employ iconic rather than textual representations, multi-font typographic-style text, and other artwork. Transient pop-up menus decoupled the information that could be accessed from the physical limitations of the screen. Multiple windows transferred even more control to users' hands, allowing them to switch between tasks and visual contexts at will. Color, motion, audio, all provided more realism and a richer space of sensory cues. As these technical advances escalated, so did the programming problems of user interfaces, problems that are only now being tamed by suitable programming environments.

Meanwhile, users began to expect each application program to have its own interface, not just the operating system as before. The prospect of interactive editing created the need for editing interfaces, first as powerful command languages for specifying text transformations (for teletype-style editing), then as fully-interactive screen editors (for cursor-addressable VDUs), and eventually as on-screen typographical editors that allowed users to manipulate a typeset image—often with built-in graphical editing facilities for illustrations as well. The growing use of text editors or word processors by non-programming personnel emphasized the importance of interface design. Following editors, other programs began to acquire individual interfaces. The invention of the spreadsheet calculator provided an great spur to interface design, for it became immediately apparent that vast power could be gained from reactive, screen-oriented interfaces.

Current user interface technology

Command-driven interfaces employ an artificial, imperative linguistic medium to allow users to control the machine through incremental interactions. The system is a passive slave awaiting orders; no attempt is made to guide or help users. On receiving an order it executes it and then awaits the next command. The UNIX system interface, called “shell,” is a typical example (Figure 1). Teletype-like, it makes no use of the cursor control features provided by VDUs. With the single exception of the character-erase and line-erase characters, the screen is treated like a long roll of paper. As further commands are entered, old information scrolls irretrievably off the screen.



```
Unix: pwd  
/home/saul/encyclopedia-article  
Unix: ls  
figures          ians-comments    sauls-comments  
working-draft  
Unix: mail  
No mail  
Unix:
```

Figure 1. A Unix command screen, showing user-typed commands in italics and prompts underlined. In this sequential dialogue, the user prints the current location in the file directory hierarchy, requests a listing of files, and checks for mail.

Menu interfaces, in contrast, explicitly reveal all possible options to the user, by analogy to a restaurant which presents the diner with an explicit list of choices. There are many different ways of arranging menus, including *direct access* menus, which show all possible choices on a single display, perhaps as a panel of buttons, and *taxonomic* menus, which classify the domain hierarchically and allow the user to navigate through it. In many circumstances it is not necessary for a menu to remain permanently visible on the

display screen, and it can be “popped up” on the screen when required. Typically a mouse button is depressed to display it, and the menu is painted on the screen near the cursor position (at the focus of visual attention). When the button is released the menu disappears and the hole left by it is automatically repaired. Menu selection is achieved by pointing at the desired item with the mouse, and indicated visually by shading that menu item. The pop-up menu is a convenient way to keep frequently used commands accessible without occupying space on the screen. Several different menus can be provided by having “buttons” on the screen which, when moused, display a menu; these are called “pull-down” (Figure 2). Normally they remain drawn only while the mouse button is pressed, but sometimes the user can move them with the mouse and post them elsewhere on the screen—“tear-off” menus.

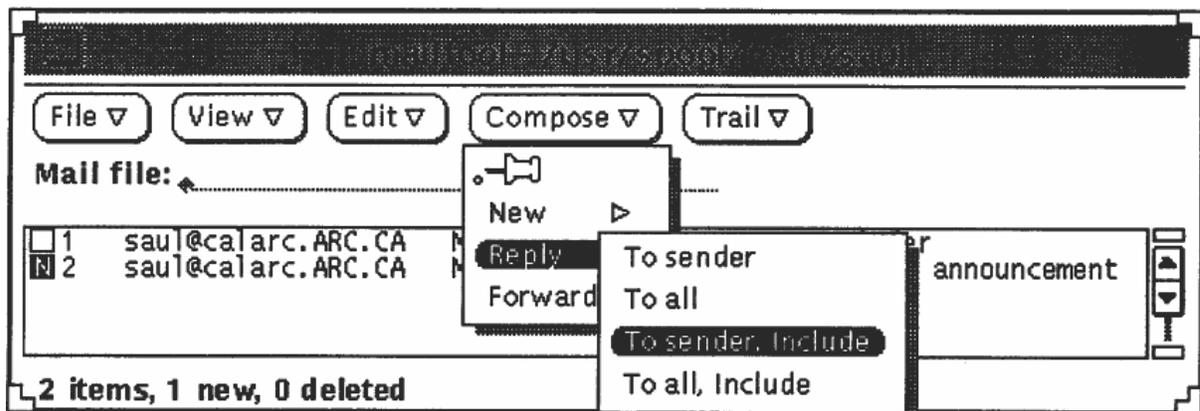


Figure 2. Sun's XView Mailtool, showing a variety of menu styles. The buttons on the top can act as both a direct access menu to execute a function immediately and as entry points to hierarchies of pull-down and “pull-right” menus. Menus can be torn off by pinning them to the display with the pin icon. The scrollable list (middle of the display) is itself a selectable menu of old and new mail.

Forms are a natural user interface medium, being widely-used tools for structuring information in conventional offices. A form is a template which, when filled in, becomes a text document (Figure 3). Either it can be viewed as a document in itself, or the filled slots can be regarded as a collection of entries in a database. This dual nature gives the form an important advantage over other ways of recording information.

A conventional paper form can:

- display information as a structured and stereotyped document;
- collect information, and permit its modification;
- store and retrieve information as records in a database;
- transfer information as messages.

Each role can be expanded within a computer forms system. First, the medium of display is not fixed. Viewed as a document, the form may be displayed, printed, typeset, even spoken over the telephone. Different text templates can be used to present different views of the same information. Only the necessary portions of the form need be disclosed to the viewer. Second, there is a similar variety of media for information collection: hand-printing on a tablet, VDU interaction, speech input, off-line data entry. Third, the

relational model of databases is the most natural for storing form-based information—each instance of the form representing a single tuple of a relation. As with a spreadsheet, implicit references to other database entries allow fields to be filled automatically (e.g. to look up an address associated with a user-supplied name field), or defined to contain the result of operations on other fields (e.g. a “total”). Fourth, mailing of forms may be expedited by transmitting only the form identification and the contents of the fields; the text template can be regenerated by the recipient from a master copy. Finally, the act of entering values may cause side effects—filling in a patient record may trigger the automatic mailing of a bill.

Dental Record	
Name: Marna Covens	
Address: 91 Roundtree Cr NW	Phone: (403) 296-1111
City: Calgary, Alberta	Postal Code: T2L 1G6
Last seen: Mar 16, 1987	
Next appointment: Dec 2, 1991	<input type="button" value="re-schedule appointment"/>
System Notes:	
1. Unpaid balance of \$127.50	<input type="button" value="review balance"/>
2. Last mail returned as "Moved to new address"	

Figure 3. A simple computerized form. Acting on a call from a dental patient, the secretary supplies the patient's name and desired appointment time. All other fields are retrieved and displayed by the system. The appointment date is automatically entered into the dentist's schedule form, checking for conflicts. Hidden parts of the form are raised by pressing the appropriate button.

Natural language seems an attractive proposition for user interfaces. However, despite the existence of some sophisticated example systems, it has not achieved the maturity of the other techniques discussed here, and its future is still uncertain. It offers the potential of very high expressiveness, combined with ease of use and familiarity for all. Users feel comfortable and are practiced in its use, and need no special training. Set against this is verbosity and the difficulty for many people of typing. Although speaking natural language seems attractive, existing speech recognition systems are highly limited research projects. A serious problem with natural language interfaces is that they only implement a rather *unnatural* subset of the language. It is very easy for users accidentally to step outside their limited domain and context. There is no warning of the boundaries of the system, and the only way to learn them is by trial and error. It is very hard to constrain one's word and syntax usage according to prespecified rules.

Direct manipulation interfaces behave as though the interaction were with a real-world object rather than an abstract system—video games provide an excellent illustration. By de-emphasizing verbs (commands) in favor of nouns (objects), communication can be made more concrete because the objects can be represented pictorially as *icons* rather than linguistic tokens. Language syntax and symbolic references are replaced by direct manipulation of the object of interest (usually with a

mouse). Direct manipulation systems map the interface structure on to some facet of the real world—a metaphor for interaction—and proceed to simulate this. A “soft machine” is an interface that employs a literal metaphor to simulate directly all features of a physical machine (such as a hand-held calculator). By choosing a metaphor familiar to the user and appropriate to the task, rich interfaces can be learnt with little training. Many modern electronic office systems follow a “desktop” metaphor through simulation of document windows, in/out trays, rolodexes, folders, filing cabinets, and trashcans (Figure 4). Abstract operations on these objects are accessed through pop-up menus. By allowing only “legal” manipulations of the object, and by altering the menu so that only appropriate abstract actions are included, errors of syntax can be avoided altogether.

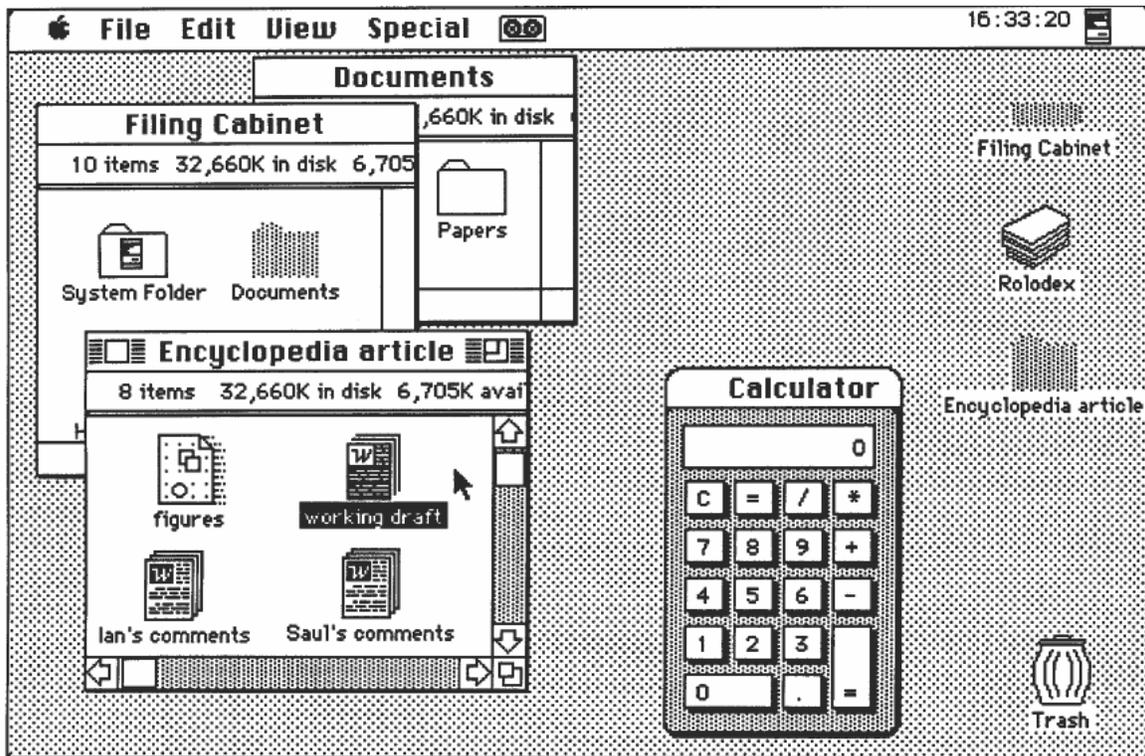


Figure 4. The Apple Macintosh desktop. Folders and documents are visible as graphical icons. Open folders are displayed in the three overlapping windows. Also visible is a calculator (a soft machine), a Rolodex icon for accessing a phone list, and a trash can for discarding unwanted items. The bar on the top of the screen includes entry points to a pull-down menu system and status information. The “Filing Cabinet” icon actually represents the internal hard disc.

New paradigms for interfaces

New paradigms for interfaces are continually being developed. For example:

Hypertext extends the notion of a document beyond sequential text by allowing complex interwoven structures to be created and manipulated by linking text fragments. The fundamental idea is simple: links can be added anywhere in the text database which, when followed, will transport the reader to another location. Associating types with links extends the power for enhancing semi-structured access to a document’s contents, with instant availability of related information; rich searching and indexing facilities; selective

and personal in-depth explorations; annotations comprising definitions, footnotes and asides; and convenient opportunities for activities such as adding personal annotations and place-marking.

Hypertext becomes *hypermedia* when any media form can be used and linked into the document. An author's point may be annotated with an instantly accessible image, sound track, or video clip. Sometimes active sections may be incorporated into otherwise passive documents to permit user interaction. When this ability is added, hypermedia becomes a rich new metaphor for interacting with computers and file stores.

Multimedia interfaces transcend text and the stylized images seen on conventional systems. Color and three-dimensional graphics, animation, audio and video can make the interface come alive. Just as sound provides enriching feedback in the natural world, so it can enhance the user interface. Moving a file icon across a desktop may be accompanied by a dragging sound that reflects the underlying surface—the harmonic fullness of the file folder, the hollow tones of the background screen, the clanging contact with the trashcan. Similarly, animation of interface constructs can make visual objects seem to behave just as their physical counterparts do. Color, properly used, enriches the interface aesthetically and supplies the user with additional information without occupying extra screen space. Inclusion of video brings a new way of importing “real-life” data and impressions into the computer.

Groupware encompasses software applications for several users working together by promoting general communication between people. Considering the collaborative nature of most of today's work, it is surprising that the vast majority of current applications support only a single person's on-line activities. Tele-conferencing and video-conferencing bring geographically separated people together for real-time meetings. Electronic mail, with a delivery time of minutes, has proven an effective means for asynchronous communication, and augments the roles more conventionally assumed by surface mail, inter-office memos, facsimile transmission, and even quick phone calls. Advanced mail systems allow people to compose multi-media messages; specify criteria for filtering mail; and enforce a specific message exchange protocol. Bulletin boards are communal mail boxes where people can post, read and reply to messages, and connect an extended community of geographically-separated people with common interests. Groupware also promotes task-specific collaboration: multi-user applications can help groups to record brainstorming, list ideas, collaborate on documents, and even compare personal beliefs.

Cyberspace is an innovative and futuristic approach to human-computer interaction. It immerses a person's senses in a three-dimensional simulated virtual world. Seeing the world in a stereoscopic head-mounted display which has a screen for each eye, one moves through it by head and body gestures. Motion sensors pick up and translate real movements to virtual ones, and the view is adjusted accordingly. Users interact with the simulated world through a data-glove or data-suit that allows them to grasp and manipulate the virtual objects they see. They hear sounds through a 3-D audio display. The effect, although still primitive, is to exist and interact within a virtual reality—cyberspace.

Designing user interfaces

Designing and building a viable user interface requires creativity, knowledge of design guidelines, suitable tools, and techniques of evaluation.

Design guidelines are distilled from empirical studies and practitioners' experience. The golden rule is "know the user," which includes familiarity with the task, environment, personal capabilities and limitations, and likely reaction to the system. Other guidelines range from common sense motherhoods ("provide good feedback") to quite specific rules ("do not use the color blue for critical data"). Some computer vendors even provide *style sheets* recommending a generic "look and feel" for interfaces to follow. Guidelines and style sheets require informed interpretation, and do not constitute recipes that should be blindly adhered to. Nevertheless, they serve to indicate what might be considered, and what choices other designers have found useful.

User interface toolkits encapsulate standard interface constructs (such as windows, menus, control panels, dialogue boxes) into a subroutine package for programmers. User interfaces are notoriously time-consuming to build, and as their complexity increases so must the sophistication of the tools used to develop them. Toolkits not only help the designer create interfaces rapidly, but also promote consistency in style between applications. Many vendors now endorse standard toolkits (for ease of portability across various hardware platforms), and a standard "look and feel" (for consumer acceptance). The most notable is X-windows, a portable window system. Consortia such as the Open Software Foundation and Unix International are promoting the Motif and Open Look toolkits respectively for developing applications within X-windows. In contrast, the Apple Macintosh has a high quality but proprietary user interface toolkit, and Apple has not hesitated to sue vendors who copy their "look and feel."

User interface management systems (UIMS) decouple application programs from the appearance of their interface, the two being linked by some intermediary abstract specification. The UIMS manages interface presentation and user interaction at runtime. This architecture allows the interface to be changed without altering the application program. The application can be built independently, and the interface can undergo iterations of rapid prototyping and user testing until a satisfactory design is found. Some systems even allow interactive selection and layout of the user interface building blocks through "interface builders." Examples of commercial UIMSs are *MacApp* for the Macintosh and *Open Dialog* for the Apollo.

Interface evaluation is necessary if interfaces are to be improved by iterative design and testing. It is difficult for a designer to discover if the interface built is actually a good one. Intuition, while valuable, can seriously mislead because the designer is often quite dissimilar to the targetted user. More objective evaluation requires watching (perhaps on videotape) the intended user trying out the system, and noting where the design fails. This should happen early in the design process, possibly through prototypes, mockups, or even paper walk-throughs. More formal methods of evaluation involve collecting data on user activity, statistical testing, and protocol analysis.

REFERENCES

1983. Card, S.K., Moran, T.P. and Newell, A. *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.
1984. Smith, S.L. and Mosier, J.N. *Design guidelines for user-system interface software* Bedford, MA: Mitre Corporation.
1986. Norman, D.A. and Draper, S.W. (Editors) *User centered system design—new perspectives on human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.
1987. Baecker, R.M. and Buxton, W.A.S. (Editors) *Readings in human-computer interaction*. Los Altos, California: Morgan Kaufmann.
1987. Carroll, J.M. (Editor) *Interfacing thought: Cognitive aspects of human-computer interaction*. Cambridge, MA: MIT Press.
1987. Shneiderman, B. *Designing the user interface*. Reading, MA: Addison-Wesley.
- In press. Thimbleby, H. *The user interface design book*. Reading, MA: Addison-Wesley.

Ian H. Witten
Department of Computer Science,
University of Calgary,
Calgary, Canada T2N 1N4

Saul Greenberg
Advanced Technologies
Alberta Research Council
Calgary, Canada T2E 7H7