LBL-23731
Preprint

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA, BERKELEY

Information and Computing Sciences Division

To be submitted for publication

Prologue to the Standardization of User-System Interfaces for Office Systems

D.F. Stevens

July 1987

Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

DISCLAIMER

PROLOGUE TO THE STANDARDIZATION OF USER-SYSTEM INTERFACES FOR OFFICE SYSTEMS*

David F. Stevens
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

Summary: The total user-system interface for computer office systems contains sub-interfaces of three fundamentally different types: wholly direct, semi-direct, and wholly indirect. Recent standardization activity is concentrated in the area of the semi-direct sub-interface, which is where the bulk of the perceived interface resides.

Introduction

The standardization of the user-system interface for office systems is an idea whose time seems to have burst upon us. At least half a dozen projects have been recently begun at both the national and international levels. Within the US alone, there is activity in two ANSI Accredited Standards Committees (T1 and X3) and in the Human Factors Society. Much of the earlier work on interface standards, such as the QWERTY and Dvorak keyboards, floppy disks, and character set encoding, can be characterized as being directed at the *mechanical* or *mechanistic* aspects of the interface; the new standards, by contrast, are directed more at the *psychological* aspects of the interface [1]. To see why this shift in emphasis is appropriate, we shall consider the nature of the user-system interface, with a view towards understanding those parts of it that do the most to determine its usability, friendliness, and acceptability.

The Nature of the Interface

Modern office systems are designed to conceal much of the computing machinery that underlies their surface appearance. This spares the user the burden of learning bit-level (or even procedural level) system details, but he still must deal with a portion of the higher level system internal constructs and actions involved in his applications, plus the full repertoire of symbols through which the system attempts to communicate with the user, and the physical devices associated with his workstation. The user-system interface thus contains three different kinds of objects -- physical devices, system internal constructs, and

* This work was supported by the U.S. Department of Energy under contract No. DE-AC03-76SF00098.

symbols; the actions and operations -- both actual and symbolic -- permitted on and with the various objects, the interpretations applied to objects and actions, and the vocabulary used to define and describe them. It -- the interface -- is thus an extensive and diverse collection of physical and logical entities. It may be "easy to learn" or "user friendly" or extremely desirable in some other fashion, but it is almost certainly not straightforward. There are numerous reasons for this, including the following:

- The repertoire of operations and actions available ranges from very low level (tilting a VDU screen; executing a bit-level machine instruction) to very high level (paginating an entire document, with automatic placement of footnotes, (re)numbering of pages, and update of the index, all as a single operation).

- It is generally impossible to predict the complexity of the operation performed from the complexity of the action required to perform it. Simple operations may require relatively complex sequences of actions (deleting a document in a protective icon-based system can involve selecting the document with a mouse, pressing a function key or selecting a menu option, and reconfirming the intent), while some extremely complex actions can be accomplished rather simply (the pagination example given above can be invoked by the selection of a single menu option).

- The set of symbols available can be limited to the graphics that represent characters input through a keyboard, or can be expanded to include pictorial objects of arbitrary complexity.

- Many objects, symbols, and actions have context-sensitive interpretations.

The collection of objects and actions involved in even a restricted subset of the applications possible on a modern system is both extensive and disparate, and yet the user must maintain familiarity and competence with them all if he is to use his system effectively. He must learn not only how to deal individually with fundamentally different kinds of objects, but also how they interact with each other. The specification and accomplishment of most tasks proceeds more or less as follows:

1: The user translates his real task (the printing of a document, for instance) into the actions he must perform. This may either be done as a whole, before he begins the actual operation, or as the operation progresses, with each step informing its successors. It requires an understanding of the capabilities of the system at the application level, but does not necessarily require any knowledge of the inner mechanisms of the computational tasks that go to create the application.

2: The user performs the ritual necessary to initiate the task. In a command-driven system, this means entering a command; in a menu-driven system, selecting the right choice (or sequence of choices) in a menu; in a window-and-icon system, selecting an icon and manipulating it. In all

cases, the user manipulates one or more of the physical devices in ways that are understood by the system.

3: The system interprets the changed state of the physical devices manipulated by the user, updates the screen-resident representation of the system, and interprets the changed state of the system universe as now represented by the screen. The user interprets the changing display on the VDU to monitor his (and the system's) progress.

4: From time to time, the system decides that it has enough information to change the system itself to conform to the instructions communicated by the actions of the user, or implied by the changes in the screen representation. These changes to the system are communicated to the user, occasionally through changes in the physical devices, but more usually through changes in the location and/or appearance of the symbols on the screen.

5: When enough information has been transmitted from the user to the system, the system and its devices perform the task requested.

This quick overview only hints at the true complexity of the interaction; apparently simple actions can involve many iterations of various action/interpretation sequences. For example, consider how the user "moves" a pointer with a mouse. He actually moves the mouse; the system senses the motion of the mouse and translates it into motion of the pointer on the screen; the user observes the motion of the pointer, which he uses as feedback governing his hand motion. This continues until the system senses that the pointer has moved to a significant area of the screen; it then generates the appropriate activity and changes the state of one or more of the symbols (by highlighting, for example) to reflect the new state of the system. The user interprets the change in the symbols, and reacts accordingly.

This complexity is not necessarily bad, for one of the interesting results of our early experience with modern systems is that the multitude of interfacing techniques they employ often turns out to be more well-suited to the demands of the worker and workplace than does the "simplicity" of the traditional single technique of command language. It appears that a complex but intuitive interface is preferable to one that is simple but counterintuitive .

A User-System Interface Model

We have seen that the office system user-system interface is a complex assortment of different kinds of objects that interact with the user and with each other in fairly intricate ways. The number of potential candidates for standardization is thus extremely large. We can achieve some insight into the most fruitful areas for standardization if we first categorize the elements of the interface into a relatively small number

of classes. We introduced the discussion of the nature of the interface by introducing three classes of objects: physical devices, symbols, and system internal constructs. These three types of objects suggest a natural partition of the total interface into three sub-interfaces, one dealing with each class. The nature of the interface is such that, while they are distinct, the three subinterfaces are intertwined. To see this, let us begin with the model of Figure 1, which shows the user, the three classes of objects, and the links

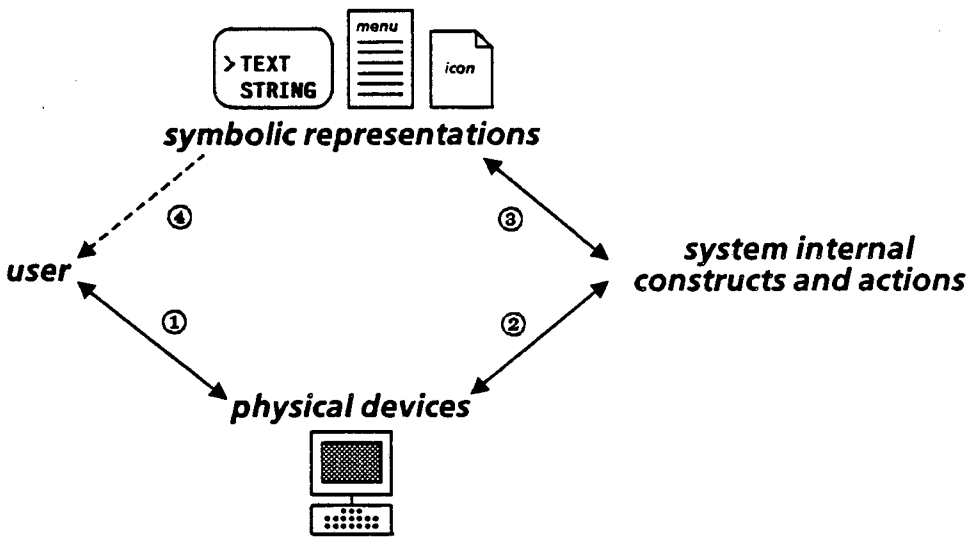

Figure 1: User-System Interface Model for Computer-Based Office Systems

between them, and recast our general operational schema (of the preceding section) in terms of the model:

1: The user translates his task into the actual sequence of actions he will use.

2: He performs the ritual necessary to initiate the task by manipulating the physical devices of the workstation, through link ①.

3: The system is informed of the user's actions through link ②, interprets them, and updates the screen-resident representation of the system (link ③). The system also interprets the changed state of the system universe as now represented by the screen (which it receives through link ③ again, but in the other direction). Either of these interpretation processes may result in changes to system structures. The user monitors his (and the system's) progress by viewing the screen (link ④). (Note that link ④ is unidirectional.)

4: After certain of the changes to system structures, the system updates the display and/or changes the state of some of the physical devices attached to the system (links ③ and ②). These actions are

communicated to the user via link ① (in the case of the physical devices) or link ④ (in the case of the symbols on the screen).

5: The system and its devices perform the task requested.

The more detailed description of movement of a pointer with a mouse, in terms of the model, proceeds as follows:

The user moves the mouse (①);
the system senses the motion of the mouse (②) and
translates it into motion of the pointer on the screen (③);
the user observes the motion of the pointer (④),
which he uses as feedback governing his hand motion.
This continues until the system senses that the pointer has moved
to a significant area of the screen (③);
it then generates the appropriate activity and changes the state of one or more of the symbols (by highlighting, for example) to reflect the new state of the system. (③ again)
The user interprets the change in the symbols (④), and reacts accordingly.

With this background, we can see that the user's interactions with the three kinds of objects are fundamentally different, leading to the definition of the three subinterfaces:

a) The *wholly direct* interface, where the user interacts with the physical devices of the system. These interactions involve both *direct experience* and *direct manipulation.* The user sees, hears, and feels the devices (direct experience, via link ①), and also moves them directly, that is, without the intervention of any other agent (direct manipulation, also via ①). The properties that apply to the wholly direct interface are the physical properties of the devices; examples are size, shape, color, location, orientation; resistance to pressure (of keys); brightness, contrast, sharpness of focus (of the VDU screen); rigidity of attachment (of keyboard to screen).

b) The *semi-direct* interface, where the user interacts with the symbols that represent system-defined objects and operations.

These interactions involve *direct experience* and *indirect manipulation.** The user sees† the

---

* It will be seen that this terminology conflicts with current usage in the human factors community. In human factors work the term *direct manipulation* refers to any part of the total interface that does not use command language (see, for example, [2]). In the model presented here, *direct manipulation* refers only to those parts of the interface that the user manipulates directly, i.e., without intermediation, and *indirect manipulation* is used whenever intermediation is required. Whether one uses

symbols (direct experience, via ④), but can move or change them only indirectly, through the manipulation of physical devices and interpretation of the system (via the long path ① + ② + ③). The properties that apply to the semi-direct interface are not easily characterized. They include distinguishability between different symbols, the mnemonic value of the symbols, the ease with which a desired symbol can be selected, the perceived directness of the coupling between manipulations of physical devices and the resulting observable effects on symbols, the complexity of the sequence of actions that must be performed to achieve a desired result.

c) The w*holly indirect* interface, where the user interacts with internal system-defined objects and actions.

These interactions involve *indirect experience* and *indirect manipulation*. The user does not see system-defined objects directly, but only the symbols that represent them (his experience of system internal constructs and actions is thus via the path ③ + ④). The user must infer the state of the actual objects from the observed state of the symbols. The user also has no direct means of manipulating system-defined objects, and can manipulate them only by manipulating the physical devices associated with the system, either directly (① + ②) or via manipulation of objects in the semi-direct interface (① + ② + ③ + ④ + ③). The properties that apply to the indirect interface are those that relate to the kinds of objects that are definable within the system and the repertoire of actions that the system permits; of most concern for modern office systems is the depth of detailed knowledge the user must possess to use the system effectively.

The Importance of the Semi-Direct Interface

The semi-direct interface contains the symbols that the user actually sees when working with the system, whether they be character strings or pictorial representations; it also specifies the operations on those symbols that are available to the user, and, with the cooperation of the wholly direct interface, governs how easy it is for him to perform those operations. It is primarily the semi-direct interface that defines the universe of discourse within which the user and the system cooperate to accomplish the desired tasks. The semi-direct interface is thus the critical interface for determining system usability. (The other two sub-interfaces can be *dis*satisfiers, of course, making the system unpleasant to use, but they do not have the

---

command language, joystick, mouse, or touchscreen for manipulation of the symbolic objects of the semi-direct interface, intermediation by physical devices is necessary; therefore, *indirect manipulation* is the more accurate term. (The author has proposed elsewhere [3] that the term *asyntactic manipulation* be used instead of *direct manipulation* to indicate the absence of command language.)

† Direct experience of objects in the semi-direct interface is not limited to the sense of sight, but that is the kind most often encountered. The exposition is greatly simplified if it is expressed in terms of sight alone.

power of the semi-direct interface to actively promote satisfaction.) This assertion is supported more fully in the paragraphs that follow.

The symbol becomes the object, first in language, then in thought.

In early interactive systems, the only symbols available were character strings, so that the VDU screen functioned as a kind of electronic coding sheet rather than a fundamentally different kind of device. Systems doing office work (document editing, for instance) looked more like computing systems than like office systems. The vocabulary, both for data elements and for the operations permissible, tended to reflect the computing world rather than the office world. Since the symbols were computing-oriented words, the objects remained computing-oriented objects. One did not speak of *paginating a document* in these early systems, for instance, but of *formatting a file.*

Newer systems provide a much richer environment, both of symbols and of actions, and their users are much more attuned to the application than to the system. They often have little knowledge of the computer-level operations taking place within the system, and they replace that lack of knowledge with application-oriented descriptions of the symbolic objects they manipulate and actions they perform. Thus, instead of *initializing a file* they *open a window*; instead of *entering a command* they *pull up a menu and select an operation.* Icons are initially viewed as what they *are* (pictures on a screen), but soon become the documents, file folders, printers, and wastebaskets that they *represent.* A standard method of signalling the desire to print a document in an icon-based system is to MOVE or COPY a document icon to a printer icon. The people who use these systems do not refer to this action as MOVEing or COPYing, however; it is universally called "printing".

The symbols and actions provided, and the names applied to them, become the application metaphor.

There are at least four ways to look at what the user of a computer-based office system does:

- What the user *actually* does; i.e., what buttons he pushes, etc.
- What the user *thinks* he does; how he talks about it, to himself or to others (creating files or creating documents; viewing or listing text; formatting or pagination, etc.).
- What the *system* actually does (creating and maintaining tables, compiling and/or executing programs, defining and modifying files, etc.)
- What is accomplished; e.g., the computation or document edit performed.

The second of these viewpoints -- what the user thinks he does, and how he talks about it -- defines the application metaphor. The names given to the user-level operations by the designers of the system provide the skeleton of the metaphor. In traditional command-driven systems, these names are the names

of system commands, and run the gamut from self-evident (COPY) to completely obscure (*awk* [4]). In icon-based systems, they tend to take the name of the target application-oriented operation rather than the immediate manual or representational operation performed by the user. (The print "command" referred to above is an example of this effect.) The development of the application metaphor is not limited to individual objects or operations; it can extend to the whole of the semi-direct interface, so that the screen is now no longer the *screen* or the *environment*, but the *desktop* in some systems.

The nature and quality of the wholly indirect interface (where the user interacts with actual system constructs) is immaterial if the semi-direct interface is successful.

Because the application metaphor resides in the semi-direct interface, it is at this interface that the user actually works when using the system. In almost all systems, the wholly indirect interface is obscure, unnatural, and clumsy. The limitations of this portion of the interface require that real tasks be decomposed into system-acceptable subtasks. The one-time engineering calculation is the classic example of this unnatural decomposition. In early systems, the user could not just state his problem and receive the answer; he had to be painfully aware of such *system* activities as program entry, compilation, linking, and loading, as well as program execution, and he generally had to request several of those incidental tasks explicitly. In modern systems, many of the system-induced decompositions are hidden from the user, so that he need not be aware of the necessity to "open" files or to allocate space, for example. As long as the user does not have to deal directly with the awkwardnesses of system internals that characterize the wholly indirect interface, they are immaterial.

The wholly direct interface (where the user interacts with the physical devices) is successful only if manipulations accomplished in the semi-direct interface are perceived as direct.

The principal function of the physical devices of the wholly direct interface is to provide the user with the means to observe, create, and manipulate the symbolic objects of the semi-direct interface. For traditional command-language interfaces, this reduces to the creation and display of character strings, but for modern interfaces it includes a wide variety of graphical input and output in addition to conventional keyboarding. Applications that require great precision in positioning a cursor or stylus cannot be implemented in systems that do not exhibit close coupling between the physical controlling device and the displayed object. The more precision that is demanded in the movement of an object on the screen, the more direct the connection between the controlling device and the object must appear. If there are delays, so that the user must wait for the system to "catch up", or if system motion is visibly quantified when user motion is continuous, the difficulty of the action is significantly magnified, and the perceived success of the interface is reduced.

This close coupling to the semi-direct interface is only a *necessary* condition for success of the wholly direct interface; it is not, in general, *sufficient*. A closely-coupled mouse, for example, would not be as successful as a data tablet in an application that needs direct positioning capability (as distinct from relative motion).

The semi-direct interface provides the bulk of the perceived interface between the user and the system.

After one becomes accustomed to a well-designed system, it requires conscious thought to separate the real actions of the user (movement of the mouse, for instance) from the effects produced on the screen (movement of the pointer with which the mouse is coupled). Thus, although manipulations of symbols in the semi-direct interface require the use of wholly direct interactions (i.e., manipulations of physical objects), these physical actions cease to have meaning in their own right and become merely the means by which the user operates in the semi-direct interface. The semi-direct interface also contains the application metaphor, and provides an application-oriented layer of actions and objects that insulate the user from some of the more obscure computational details of the underlying system. The user's attention is concentrated in the semi-direct interface, and the language he uses to describe his work is the language of the semi-direct interface; except in disruptive special circumstances, the total perceived interface has been reduced to the semi-direct interface.

The real world is what's on the screen.

This is essentially a colloquial restatement of the preceding assertion. It is the office system version of a well-known result from general systems theory [5, 6], and hence needs no further justification.

Combining these assertions, we see that

1) The semi-direct interface hides the awkwardnesses of the wholly indirect interface;
2) Operation in the semi-direct interface is the *raison d'être* of the wholly direct interface;
3) The user's attention is concentrated in the semi-direct interface; and
4) The application metaphor resides in the semi-direct interface.

From this it follows directly that:

The acceptability of a computer-based office system depends essentially upon the acceptability of its semi-direct interface.

Standardization in the Semi-Direct Interface

As we noted in the Introduction, the user-system interface standards that already exist tend to be concentrated in the wholly direct and the wholly indirect interfaces; they apply to the physical devices of the workstation or to the system itself. The new standards work proposed in [1], on the other hand, is taking place in the semi-direct interface; it is concerned with the whole repertoire of images the user sees on his display screen and the methods by which he manipulates them to achieve desired effects. This shift in emphasis is both timely and welcome. It is timely, because it recognizes that command language is no longer the only available control mechanism. The new control mechanisms and techniques, such as function keys, menus, windows, and free-hand pointing devices, demand new approaches to standardization; user-system interface standards must now consider what the user sees, and when and where he sees it, as well as the repertoire of "commands" at his fingertips. The new thrust in standardization is welcome, because, as we saw above, it is in these new areas of activity that the application metaphor resides, and therefore it is only there that standardization can be truly effective at simplifying the life of the user.

While it is a truism that standards work proceeds rather slowly, it would be foolish to expect that nothing will come of these efforts. On the contrary, there is every indication that the work will go forward, and will produce standards that affect most information workers quite directly.

Acknowledgement

Earlier versions of this article have been reviewed by members of the staff of the Lawrence Berkeley Laboratory, and by Task Group 9 of ANSI Accredited Technical Committee X3V1. The author is indebted all of these people, especially Fran Blackman of X3V1/TG9 and Ken Wiley of LBL, for numerous suggestions that have greatly improved the clarity and readability of the text; any obscurities or blunders that remain are the author's alone. It is also emphasized that the opinions expressed here are those of the author, and do not necessarily represent the official position of any organization mentioned in the text.

References

1. ISO document ISO/TC 97/SC 18 N 1094, *Programme of Work on User-System Interface*, June 22, 1987, contains three annexes describing new work items to be voted on this summer (1987) by the ISO community: User Guidance, Dialog Interaction, and Names for objects and actions commonly used in text and office systems. The aim of these standards, as expressed in the first annex, is that "sufficient standardization should take place to ensure that users of totally different text and office systems have an expectation of what they should find, and how they should find it."

2. Shneiderman, B. *Designing the User Interface*, Addison Wesley, 1986, Chapter 5.

3. Stevens, D. *Proposal to avoid the term "direct manipulation" in discussions of the human-computer interface.* (Submission to Task Groups 1 and 9 of standards committee X3V1.)

4. Aho, A. *et al.* *awk – A pattern scanning and processing language.* Bell Laboratories, 1978.

5. "People's knowledge of events is limited to the ways they are represented by [the] machine and by the ways in which they can alter these machine representations. . . . People vary what the program lets them vary and ignore everything else." Weick, K. Cosmos vs Chaos: Sense and Nonsense in Electronic Contexts, *Organizational Dynamics*, Autumn, 1985, pp. 56 & 57.

6. "Things *are* what they are reported to be." (Emphasis added.) Gall, J. *Systemantics*, Quadrangle, 1975, p. 39.

LAWRENCE BERKELEY LABORATORY
TECHNICAL INFORMATION DEPARTMENT
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720