Users' Models

Robert F. Sproull
Sutherland, Sproull, and Associates, Inc. and
Carnegie-Mellon University

Abstract

Users form mental models of how to interact with a computer, or with any machine. When a problem must be solved using the machine, the model is used to plan a series of steps that will achieve the goal. As the steps are executed, the model becomes a basis for the user's expectations about the behavior of the machine. If something goes wrong, the model is used again for problem-solving to help recover. The model thus forms the basis for reasoning about the machine.

Underlying every user interface is a user model designed by the application programmer—the model the programmer expects every user to grasp in order to operate the application. The model comprises a set of *objects* and *operations* on the objects. In a simple text editor, the objects may be characters, words, lines, and paragraphs; the operations may be insert, delete, copy, move, and so on. The model concerns only the conceptual aspects of the user interface, not the details of how information is displayed, of command-language design, of which buttons to push, or of error handling.

Designers of interactive systems attempt to "design the user model," the model of the system that the designer wishes the user to form. Little is known about the design of user models. Precise imitation of real-world machines (simulation) is only rarely appropriate. Some designers devise models that are analogous to real-world machines with which the user is familiar, but that include extensions to exploit the flexibility of the computer. In some areas, especially in graphical applications, the user may have no experience with existing methods for solving the problem, and the user model must then be carefully designed and explained. In these cases, "intuitive" user models are favored, but how is intuition to be assessed, and can a single model be equally intuitive to all users?

Rapporteurs: Mrs. M. Hindmarsh
Mr. D.H. Mundy

Introduction

The model a user forms of an interactive system becomes the basis for a user's ability to use the system purposefully, to achieve some goal. Based on his model, a user will formulate and answer many questions: can the system accomplish a particular task, what major steps will be required, will the results be in a useful form, what information will be retained in the system for later use, and so on. In computerized interactive systems, the user's model is especially useful in determining how a program or tool can be used as *part* of a solution to a problem that the tool alone cannot solve.

It is important to distinguish the user's model from other parts of the user interface, including the command language and interactive devices such as displays and keyboards. Even the form in which the model is presented to the user on a display may not be vital to the user's model; for example, a clock could be displayed in a variety of ways, all of which are consistent with a user's view of how a clock should behave. The parts of the user interface that go beyond the model are important in the design of an interactive system, but do not figure as strongly in the user's formulating methods for using the system.

Elements of a user's model

A user's model can be characterized as a set of *objects* and some *operations* on the objects. Both objects and operations may be viewed in the abstract, rather than as concrete representations, commands, keystrokes, and the like. For example, consider a modern electronic wristwatch. The user's model can be described as follows:

> The watch comprises three objects: the current time, an alarm time, and an elapsed-time chronometer. The current time is kept to the second, and includes the day of the month and the month of the year. The chronometer is recorded to 1/100 second, the alarm time to a minute. Whenever the current time matches the alarm time and the alarm is activated, an alarm signal sounds.
>
> There are several operations on these objects. All three objects may be viewed; the normal state of the watch shows the current time continuously. The current time may be changed. The alarm time may be changed and the alarm may be activated or deactivated. When the alarm signal sounds, it may be turned off. The chronometer may be reset, started, or stopped.

Although this description of the user's model omits all mention of the command language and of display mechanisms, a user can form a good idea of what the watch can be used for. There are, of course, situations where the user's model alone is not sufficiently informative. For example, can the time be read in the dark or at distances of ten feet? Or can the commands to start and stop the chronometer be invoked fast enough to time a foot race?

One reason for distinguishing the user's model from the rest of the user interface is that there can often be entirely separate user interfaces to the same user model. Consider, as a fairly instructive example, the user's model of a bank account. The most important object in the model is the account balance, although cash and checks also figure in the use of a bank account. The operations are those of deposit

and withdrawal, subject to the restriction that the account balance must be non-negative (though some banks allow balances to go negative). Some banks will offer several different user interfaces to accounts that use this model. One such interface uses bank tellers to execute transactions; the customer visits the bank to perform each operation. Banks will also execute most transactions by mail. Many banks now have "automatic teller machines" that will do many of the tasks of the human teller. And some banks have "bank-by-phone" facilities that allow transactions to be commanded by telephone, sometimes by speaking with a teller, and sometimes by giving commands using the telephone's tone generator as an input device and computer-generated voice for feedback. The user's model of a bank account is the same for all of these interfaces, but each interface has distinct ways to express commands, to receive confirmation, to handle errors, and so on.

Sometimes the user interface becomes sufficiently complex that in addition to a model of the application, a user must form a secondary model of the behavior of various objects in the interface, such as menus, windows, and icons. We will not discuss this problem further here.

Describing the user's model

Descriptions of a user's model are often an implicit part of the documentation of a computer program. These descriptions usually involve both *structural* and *behavioral* components, both of which contribute the the user's understanding of the model. The structural and behavioral elements are closely related to the objects and operations in the model.

Perhaps the simplest example of a structural component is the notion of *file*, or more frequently *text file*, a sequence of characters. The structural notion of a sequence makes it immediately clear what happens when characters are "inserted" or "deleted" from the sequence—the sequence lengthens or shortens to accommodate the insertion or deletion. Unfortunately, when such a text file is actually displayed to a user, it is broken into lines, which do not follow naturally from the notion of sequence: the line breaks are not part of the sequence. Then too, carriage-control or formatting characters can be inserted in the sequence, which cause seemingly hidden effects such as tabulation and line breaks. Thus the pure notion of sequence may be obscured somewhat by these enhancements. There are alternatives to the sequence model: some text editors work on two-dimensional files, with a number of rows of text, each of which contains a fixed number of characters. This model is more like that of paper in a typewriter and accommodates line breaks and tabulation easily, but the notions of "insert" and "delete" now become clouded by the interaction of rows and characters: for example, inserting a character is a different operation than inserting a row.

Although structural descriptions of models are useful, behavioral descriptions are far more common, perhaps because we habitually explain to someone *how* a machine operates or *how* to make it do certain tasks. For example, we explain that the automatic teller machine behaves "just like a teller," or that a certain document-preparation system behaves "just like a hot-metal typesetter." In these cases, we are appealing to the user's previous experience in operating similar devices. In the extreme case, of course, the computer system is a precise simulation of another machine, such as an airplane or automobile. Here the objective is to achieve identical behaviors, often with very different internal structures than the original device.

Acquiring a model

A user of an interactive system will build a model of the system by a variety of techniques. The entire model may be explained in documentation that the user reads. Or the user may be familiar with the model because he has used a similar system before or because the system simulates a device he has used. More often, the user builds a model by exploring the system, delving only into those aspects of the system that are necessary for the tasks he performs. Or the user may start out doing a few simple operations that are achieved by following step-by-step instructions presented by the system itself.

Users will acquire models by exploration even if other methods are available. For exploration to be effective, an interactive system must be designed with exploration in mind, e.g., so that a user is unlikely to invoke irreversible actions while exploring new commands. Exploration is attractive to the user because he or she needs to learn only enough about an interactive system to meet his or her needs. For example, if a text editor contains a complex pattern-matching mechanism as part of a search command, many users can defer learning about the matching mechanism until the need arises. (Of course, it is helpful if a straightforward search for a literal string is easily done, so the user need not learn about arbitrary searches in order to do a simple one.) As more of the system is explored, the user builds an increasingly complete model of it.

Although attractive, acquiring models by exploration has some serious problems. They might be summarized by "a little knowledge is a dangerous thing:" the user may have built a consistent model based on early exploration that subsequent exploration proves wrong. This possibility is quite likely, since the user is apt to encounter the more sophisticated parts of the model late in exploration.

> I found a particularly irksome example of the peril of exploration in word-processing software for a personal computer. To edit a file, the user was urged to "read in the file," edit it, and then "write it out," and users quite easily came to learn the editing commands by exploration, since changes were readily apparent on the screen. The user's model worked fine until one day the file got too large to fit in memory (more than 10 pages of typewritten text), causing a "memory full" error message. To edit large files, this software requires the user to operate on memory-sized batches, reading them from the input file, editing them, and writing them onto an output file. This method requires an entirely different user model, introduces a number of new concepts (open file, input file, output file, batch) and a number of new confusions (What if I add text at the end of the batch shown on my screen? If I work on batches, in what order will the batches appear in the output file?). Exploration here launched the user into the tarpit of an entirely new model.
>
> The notion of *file* is often difficult for those unfamiliar with computers, and exploration does little to uncover the properties or behavior of files. Files are rarely explained or explained well in documentation, and never in on-line material—the user of a computer is assumed to be familiar with files. But files embody some quite subtle concepts, which have no close real-world analogies to aid comprehension. For example, the contents of a file may change even though its name does not. When I "edit" a file, am I editing a copy of the file or the file itself? What is the purpose of the file's name, and how should names be organized (especially if the file system is

hierarchical)? Exploration of these notions is not often feasible, in part because a file is an *object* that can be explored only by performing *operations* on it, and the operations are usually distributed in many different application programs.

A model can sometimes be very hard to acquire if the user's previous experience contradicts the model. For example, it is hard to teach a draftsman about a solid-modeling system, because all of his experience deals with plane sections of objects. The information necessary or desirable on a two-dimensional drawing is very different from that for a solid model. Another example is that of a typesetter learning to use a document-formatting system based on "generic coding," in which text elements are labeled by content rather than format. For example, text will be labeled as "body" or "number 1 heading" rather than "10 on 12 point Times Roman," a conventional typesetting coding. The typesetter may become confused about the distinction between generic and concrete (or "machine") codes, or about why generic coding is being used. In systems that allow generic and concrete codes to be intermixed, the confusion is particularly severe: in effect, two different models of the document are being mixed.

Designing user models

The designer of an interactive system usually starts by "designing the user model," i.e., by developing the model of the system that he intends the user to form. This model will be presented in the documentation, in on-line help and error-handling facilities, and in the operation of the system itself.

There are several problems that the design must address:

Exploration. The user's model can be designed to encourage exploration. An important requirement is that the user be able to experiment without disastrous consequences, a property sometimes addressed by "undo" or "replay" facilities (these succeed only if the user is aware of the disaster soon enough to invoke the recovery). Menu-based systems probably encourage exploration, since the menus serve to identify the options available at every step of a dialog.

Wide audience. How knowledgeable is the user? If the user interface is designed to be used by a wide audience, the concepts in the user's model must be understandable or even familiar to that audience. Hence, for example, Star uses the concept of "document" rather than "file", and "file folder" rather than "directory" in order to appeal to existing office objects and operations. A wide audience may also require more use of concrete, less abstract, concepts in the model; again, Star avoids the abstract notion of a file in favor of the more concrete concept of a document. However, as the sophistication of the population increases, users will increasingly have prior computing experience when learning a new user's model, and the intricacy and abstraction of the models can be expected to rise.

Modification. It is usually important to design a user's model so that the system can be maintained and extended without requiring small distortions or major overhaul to the model. For example, if the notion of "printing options" is in the model from the start, adding a few new options is not difficult.

There are also several approaches to the design of the user model:

Simulation, extension, and restriction. Sometimes a user's model is designed to simulate some other real-world object: for example, a clock on the screen may behave just like a clock on the wall. More often, models are designed *by analogy* with a real-world object, where the analogy is intended to help the user learn the model but the model is allowed to go beyond the original. A word-processing program may be "like a typewriter," but it presumably is different from a typewriter in certain ways or it would actually *be* a typewriter. This can be troublesome to a user: just how much of the analogy holds? There is danger in drawing too strong parallels where none exist. For example, the "automatic teller" is in fact a severely restricted form of the real teller; the one I use has only two denominations of currency, and therefore limits the amount of a withdrawal I can make.

Tools or closed machines. There are two fundamentally different approaches to designing interactive systems: as fixed-function machines or as tools that work together with other tools to accomplish a range of tasks. We are used to machines with fixed functions: an oven, a calculator, and a watch are examples. The machine must have a command for everything it can do, and it is not designed to work with other machines. By contrast, tools are intended to be used in combinations to achieve a range of results that might be unachievable by a single machine, or at the very least the single machine would be extremely complex. For example, a text editor is a tool that may be used in conjunction with one or more electronic mail programs, document compilers, programming environments, and other applications. Generally, interactive systems tend to favor a tool approach.

Gedanken experiments. Often, in designing a user's model, it is helpful to have a collection of concrete examples of chores that the program must do. For a document-preparation system, the collection may include sample pages that the system must generate; it may include copy marked with formatting or editorial changes that must be made; it may include sample operations (e.g., renumber the sections of a chapter, or change the form of all bibliographic references). This collection is a concrete form of a requirements specification, which the designer can use to analyze alternative designs.

The designers of Star have written an excellent article describing how the "desktop" model was fashioned as an extension of conventional office practices familiar to office workers [4]. They also give an example of the kind of problem that arises when a familiar model is extended with new concepts: what happens to an *electronic* document when it is printed on a laser printer? Does a printer *transform* a document from electronic to paper form (i.e., does the electronic form disappear when the document is printed)? Or does a printer *copy* an electronic document onto paper? These questions arise only because the electronic document is not identical to a physical document, but merely a close analog.

User models of drawings

To illustrate some of the fundamental choices that a designer must make when constructing a user's model, we shall describe possible approaches to an interactive drawing program. There are two basic approaches to the user's model: a *painting* model, and a *geometric* model. These two models are compared and contrasted below; finally, a third model is presented which combines the first two.

Painting. In its pure form, this model provides analogs of brush, paint, and paper with which the user prepares a drawing. A graphical input device is used to steer the brush, a choice of different brush shapes is provided (e.g., round, square, straight, oblique), and paints of various colors are available. Wherever a brush deposits paint, the new paint obscures any paint previously deposited at the same spot. A particularly important paint color is "paper color," which may be brushed on to erase previously applied paint. This model behaves just like real-world painting.

The model is usually embellished with a set of tools to ease the creation of certain kinds of images. Some of these tools correspond to illustrators' tools in the real world, e.g., an air brush, or a "transfer sheet" of previously prepared images that can be transferred onto the paper (these include text characters, special symbols, textured patterns, etc.). Other aids are attempts to improve on real-world tools, e.g., a mechanism for moving the brush along a straight line between two points so as to draw a perfectly straight line, or a mechanism to fit a smooth curve through a set of points. These aids are really geometric in nature, but they are cast as tools for controlling the brush trajectory in order to fit into the painting model. Tools for copying, cutting, and pasting parts of the image may also be provided; these usually depend on selecting a rectangular portion of the image to be acted upon, although more complex shapes are possible. In any case, selection identifies a *region* of the paper, not the objects that appear to be drawn on the paper (because the model has no concept of drawn objects, only of paint that has been laid down). Another common embellishment introduces transparent paints as well as opaque ones; as transparent paint is deposited, it mixes with any previously-applied paint. To give more flexibility, the user can be allowed to paint on more than one "overlay," and the overlays may be stacked one atop the other to achieve the final result. This technique is used in conventional animation to allow part of an image to be moved with respect to the rest; it offers the same flexibility in its computer analog.

The principal strengths of the painting model are that everyone is familiar with it and that it allows arbitrary images to be constructed by sufficiently precise application of paint. The model has a number of drawbacks, however:

> Practical limitations on implementations detract seriously from the model. The problem is that the only representation known for paint-on-paper is a raster array of color values, a bulky representation at best. Economics and performance often combine to restrict the resolution that this array provides, e.g., to 1024 by 1024 points, not enough for high quality work. Lines look jagged, text is coarse. Moreover, operations like scaling and rotation require digital signal-processing algorithms that are slow and introduce errors that are, in many cases, obnoxious to the viewer. If the raster must be scaled in order to be presented on another display or printer, these errors can appear frequently. (The gray square in Figure 1d appears perfectly uniform on the display, but scaling it to printer resolution introduces visible errors.)
>
> Although structure may be apparent in the drawing, the painting program cannot use it. The notions of line, box, curve, or circle do not exist in the model. Thus if we want to erase the horizontal line in Figure 1a, we paint a new line with paper-colored paint, obtaining Figure 1b. But we usually want to obtain Figure 1c, which requires us to repaint at least part of the vertical line. Lack of structure also prevents moving "joints" between lines and expecting the lines that

are attached to follow (Figures 1g and 1h). Structure is also absent in text strings, so editing Figure 1i to obtain Figure 1j requires erasing the "s" by painting over it, cutting out "or" and pasting it in further to the left. If other imagery lies under the text, moving the "or" will also move the image, probably an undesirable result.

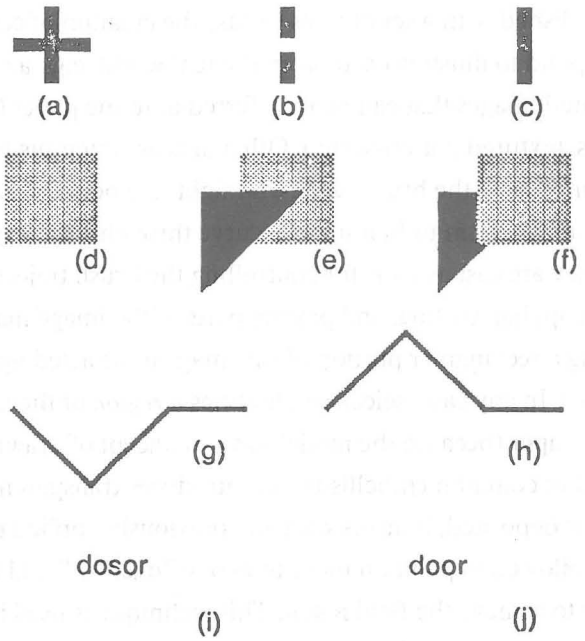

Figure 1.

Geometry. The geometric model describes a drawing as a collection of geometric objects: lines, curves, arcs, circles, polygons, polygons "filled" with a solid color, and text characters (behaviorally identical to filled polygons). Each object may have a different solid color or texture. Lines may have different widths and different end-point treatments (squared-off or rounded). Objects may be connected to one another, e.g., lines and curves at endpoints, adjacent text characters in a string.

The geometric model requires introducing the notion of *priority* to resolve the ambiguity that arises when two filled objects overlap (Figure 1e and 1f). Figure 1e is achieved by giving the triangle higher priority than the square, while Figure 1f requires the reverse.

The principal strength of the geometric model is its structural information: the entities that a user wants to manipulate in a drawing are objects in the model; they can be selected and modified. For example, a line can be deleted (i.e., a single step will transform Figure 1a into 1c). Common geometric transformations can be applied to objects quickly: translation, rotation, scaling, skewing. Objects can be grouped together to form "symbols" and manipulated as such: copied, moved, deleted, scaled, rotated, etc. The geometric primitives are relatively device-independent, so that in practice the fidelity of a drawing is limited only by the resolution of the device used to present it, not by the precision of the representation of the drawing.

The principal weakness of the geometric model is that it does not accommodate pictures, whether

derived from scanned images or drawn interactively. This is a serious limitation, because many applications require an occasional use of pictorial material (e.g., for a cover page). A related problem is that sometimes area selection is preferable to selection of geometric objects, although suitable geometric clipping algorithms can provide such a facility.

A hybrid model. It is possible to construct a hybrid model, in which both geometric and pictorial material is accommodated. The basic idea is to alter the geometric model to allow a geometric object's color to be a solid color, a texture, or a picture [5]. The pictures can be created with the painting model and the geometric objects with the geometric model.

Selecting a model. Each of these models presents the user with a *very different view of how to make an illustration.* Even without defining the command language or presentation of images on the screen, we can compare the models and try to assess their suitability for a particular application. We can consider embellishments to a model that may be necessary to meet particular needs of an application, and we can determine whether the embellishments fit with the rest of the model. We can even write the first chapter of the user's manual, where the basic concepts of the application are explained.

In many cases, the nature of the application will determine the most appropriate model. Computer-aided design (CAD) applications use geometric models: the users are familiar with geometry and the images used in these systems do not require pictorial material. By contrast, graphic arts applications such as electronic pre-press systems for preparing printing plates must handle pictures. Some of the drawbacks of the painting model are not severe in this application: device independence is rarely necessary, since images may be scanned in and out at identical resolutions or may be scanned with sufficient precision that scaling algorithm errors are not visible. But the graphic arts systems pay a price for representing the entire raster image at high resolution: the systems are expensive and can be slow.

In less demanding applications, such as illustrating technical documents or office communications, the models are not as sharply distinguished. It is notable, however, that both Star [3, 4] and LISA [6] use geometric models, in part because practical problems with the painting model can be daunting on small systems. However, if a color display were available on either of these systems, perhaps pictorial images would receive greater attention.

Conclusion

The art of designing user's models is in its infancy. Although cognitive psychologists are beginning to work on "mental models" [1], there are as yet no ways to analyze user's model designs to anticipate the user's understanding of the model or performance with it. There are, however, examples of interactive systems that have firm models underlying them: Star [3, 4] and LISA [6] are recent examples. Although many people feel these are good models, no one has figured out how to do experiments to measure their strengths and weaknesses.

A typical student of computer science is ill prepared to design interactive systems. Curricula tend to do poorly at teaching design, and usually the specification or design of a user interface is not covered at all; in some cases, students never write an interactive program. Software engineering courses emphasize the

design of abstractions (objects and operations), but the principles that make well-engineered software internals do not lead to good user interfaces. Until we are able to produce students capable of these designs, we must be content with a very limited pool of talented designers: there are probably fewer than twenty people in the world who are capable of designing modern integrated interfaces.

References

[1] D. Gentner and A.L. Stevens, eds., *Mental Models*, Lawrence Erlbaum Associates, Hillsdale, N.J., 1983.

[2] D.E. Lipke, S.R. Evans, J.K. Newlin, and R.L. Weissman, "Star Graphics: An Object-Oriented Implementation," *Computer Graphics*, 16(3):115–124, July 1982.

[3] D.C. Smith, E. Harslem, C. Irby, and R. Kimball, "The Star User Interface: An Overview," *Proc. National Computer Conference*, June 1982, pp. 7-10.

[4] D.C. Smith, C. Irby, R. Kimball, B. Verplank, and E. Harslen, "Designing the Star User Interface," *Byte*, 7(4):242–282, April 1982.

[5] J. Warnock and D.K. Wyatt, "A device-independent imaging model for use with raster devices," *Computer Graphics*, 16(3):313–319, July 1982.

[6] G. Williams, "The LISA Computer System," *Byte*, 8(2):33–50, February 1983.