CHAPTER 2

Interface Design and Evaluation—Semiotic Implications

Mihai Nadin

The Ohio State University

Knowledge and experience from several disciplines (psychology, sociology, communication theory, graphic design, and linguistics, among others) are used, or should be considered, in this still new field of human activity identified as interface design. The implicit acknowledgement of the interdisciplinary nature of interface design is symptomatic for the ability to differentiate between general and specific requirements peculiar to the use of computers by various users, as well as for understanding the need to translate this differentiation in a design appropriate to the functions desired. After an expensive and prolonged trial and error phase, researchers and industry indeed acquired a more appropriate apprehension of the complex nature of interface design and of procedures to evaluate it (before and after implementation on computers).

WHY SEMIOTICS? WHAT KIND OF SEMIOTICS?

The input from various specialized fields of knowledge made possible rapid but partial progress of interface design. While we know how to cope with complexity in particular situations, we do not know yet how to assemble successful specialized interfaces in a system which can be approached by the user at a higher level than that of each component. The ideal towards which the field is moving—emulation of human communication, with special emphasis on interactive natural language user interface—requires not only the use of paradigms and empirical findings from the various fields dealing with different aspects of human being and its social existence, but also an integrative methodology. Interdisciplinarity and integration should go together. This integrative methodology can be seen as a bridge between all the disciplines from which interface design has borrowed concepts and findings, as well as between them and computer science, whose spectacular progress made the design of interface a rather critical element of the progress of computer technology and its applications. Semiotics, a relatively new discipline, built on an impressive history of interest in its object and method, provides the bridge mentioned above.

Semiotics, the general theory and practice of signs, studies everything that is interpreted by human beings as a sign, and defines the circumstances under which interpreting something as a sign allows for its better understanding, or for an improved use of it.

People communicate among themselves, or, to use the word "communicate" loosely, with machines, using signs, some of which belong to the stabilized system of so-called natural language, others to the shared conventions of gestures, sounds, colors, or synthetic languages. People learn available sign systems and generate new signs for various purposes. Any type of knowledge is shared by means of language (the most complex sign system we know) complemented by specialized signs such as those found in mathematics, logical formalism, diagrammatic representations, and in musical or engineering notation. Scientific and speculative theories, as well as art works, are similarly expressed. Whether people regard computers as mere sophisticated tools or as machines embodying a certain degree of human intelligence (in hardware and the programs driving it), they agree that, in order to use them effectively, human computer interaction should be provided which does justice to the user and takes advantage of the computer's potential. Everything supporting this interaction makes up the entity called interface.

This interaction is not physical, as with tools designed and used before computers, but symbolic. That is, it takes place through the intermediary of shared representations, i.e., signs of intended actions, or of feedback signs designed for specific purposes. Accordingly, the underlying thesis of this chapter is that, since the user interacts with computers by means of signs, the design of interfaces can advance by applying the principles and empirical findings of semiotics. The reason for a semiotic approach lies in the very nature of interface design; therefore, requirements intrinsic to the subject should guide in identifying what kind of semiotics can be useful, as well as what can be expected from such an endeavor. Let us first identify the appropriate semiotic framework.

In order to apply semiotics, we have to adopt one of the many definitions of the sign which have been advanced. Known definitions fall into two categories:

1. Pertinent to natural language. A sign is adopted as a paradigm with the understanding that every other sign is structurally equivalent. The Swiss linguist Ferdinand de Saussure advanced the definition of sign as the unity between a *signifier* (the actual sign embodied in material form, such as words or shapes) and the *signified* (what the sign is supposed to mean). Artificial intelligence researchers are quite comfortable with this model.

2. Pertinent to local structure. Each type of sign and each sign operation can be described in a logical system. The American scientist and logician Charles S. Peirce (1839–1914), and pioneer of the computer (cf. Ketner, 1984), advanced the definition of sign as "something that stands to someone for something in some respect or capacity" (Peirce, 1932, p. 135).

Arguments in favor of one or the other definition can be produced, but, since computers are *logical* machines, a logical conception of the sign and its functioning in different contexts is more appropriate to the subject. Design in general, and interface design in particular, are not reducible to the model of natural language (or any other sign system). On the basis of Peirce's definition, this diagramatic representation (not the only one possible) can serve as an operational model.

Figure 1 should be read as saying that only the *unity* between

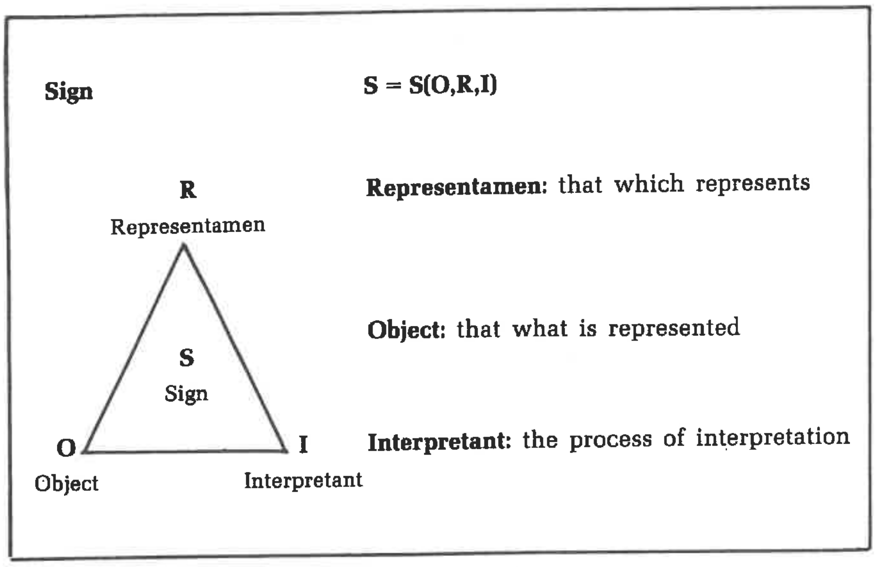

Sign
S = S(O,R,I)
R
Representamen
S
Sign
O
Object
I
Interpretant
Representamen: that which represents
Object: that what is represented
Interpretant: the process of interpretation

Figure 1. Sign definition in Peirce's semiotic.

the three components represent a sign, i.e., that signs are identified as such only through their representation, and that, where we interpret a sign, we become part of it for the time of that interpretation. Let me explain this through an example. Icons in the user interface paradigm are *representamina.* Like the icon of the file, calculator, or clipboard, every other icon represents an object with which the user is familiar from previous experience. The *representamen* of the clipboard (the drawing resembling the object clipboard that we use in school, in the office, or on other occasions) is interpreted by the user as being the computer equivalent of a stationary item. Only when the user establishes the relation between the drawing that represents the clipboard (representamen), the physical item clipboard (object), and the understanding of the function of the computer-simulated clipboard (interpretant) can we speak of constituting a sign.

The sign can be constituted according to the intention of its designer, or quite independently of it. Design of icons as part of interface design requires that we understand under which circumstances the intended or accidental user will correctly constitute the sign, i.e., interpret it according to a desired intention. Once the kind of semiotics suitable for better interface design is identified, we know what to expect from its application. Prescriptive tools will become available. Such tools help designers avoid errors when a given activity is to be modelled in a user interface supporting the most efficient interaction between computers and users. The basic prescriptive rules which semiotics provides refer to the coherence of the repertory of signs used, as well as to the consistency of sign operations applied to the repertory chosen. Second, semiotics offers tools for quantitative and qualitative evaluation through the basic semiotic principles of appropriateness of representation and stability of the dynamic sign system. The vocabulary of semiotics is less important than the awareness of how people constitute and interpret signs and use them as tools for new developments. As a metadiscipline describing signs, in particular language, by using language and other signs to constitute its own corpus of knowledge, semiotics is less a descriptive theory and more a methodology for improved interpretation and evaluation of how people communicate, represent things, express themselves, and constitute new models and theories. For this reason, it is necessary to explain how semiotics, within the adopted definition of the sign, deals with such activities. Once these explanations are provided, examples will be introduced.

SIGN AND CONTEXT

The functioning of the sign in the context it was constituted in (in our case, on the screen of a computer using a well-defined visual convention, i.e., resemblance between the representamen and the object represented) is expressed through the concept of *semiosis* (sign process). This describes how given signs preserve or change their meaning, how the change of context affects their interpretation, and how the three fundamental functions of signs are obtained (see Figure 2).

1. If, when using a sign, the emphasis is on the relation between the representamen and the object represented (which is the case of icons in user interface, in the use of pictograms in airports, etc.), we actually confine ourselves to the function of representation.

2. There are circumstances under which the emphasis is on the formal qualities of the representamen, interpreted almost independently of the relation to what it represents. This is the case with artistic expression, sometimes pursued by overzealous interface designers right up to the computer screen. (I remember the bee on the initial Intran system, flying in a strange fog while the computer was "resting." The bee turned into a cursor once the system was started, and the cursor into a hand once a graphic program was chosen.)

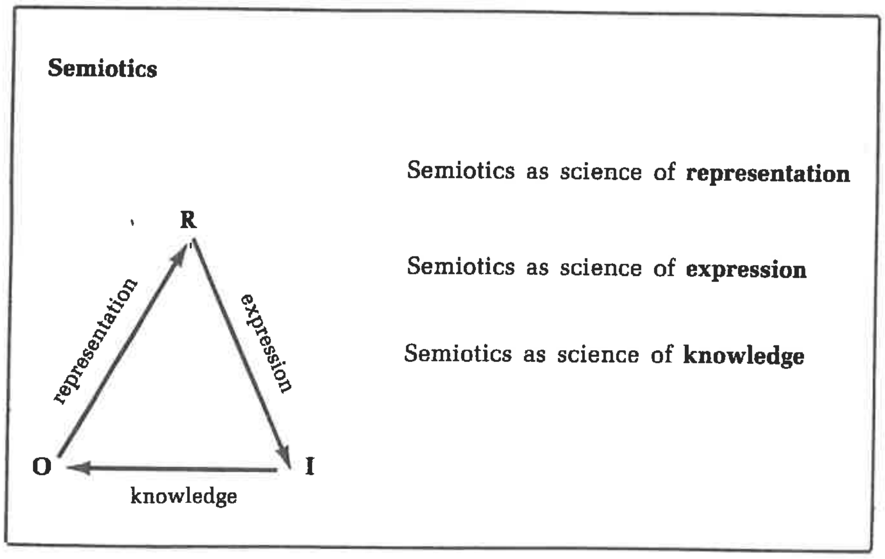

Semiotics
Semiotics as science of **representation**
Semiotics as science of **expression**
Semiotics as science of **knowledge**
R
O
I
representation
expression
knowledge

Figure 2. Sign functions.

3. If the user of a sign integrates representation and expression, and tries to derive from the sign specific or general knowledge of the object, the cognitive function is accomplished. Computational biology, astronomy "on the chip," computational physics, linguistics, etc. are examples in which a representation—let's say the DNA model—its expression (in computer graphics-adequate form), and the theoretic explanation it makes possible constitute a new body of knowledge.

Based on these concepts, we can further explain the semiotic levels at which sign processes (semioses) take place, levels that are undoubtedly familiar and important to those working in computer science (see Figure 3).

Letters, characters, words, and commands are just examples of representamina. They have a precise syntax which in turn supports the semantics (as a necessary but not sufficient condition). Even such words as "if," "the," "and," or some artificially produced sequences represent objects, i.e., concepts with a precise logical meaning. Whether images, sequences of letters or numbers, time sequences, touch sensitive areas, temperature sensitive sensors with attached functions, etc.—all have a syntax, which can be prescribed as rigidly as the system requires, on whose basis the intended semantics is built and the expected pragmatics is obtained. The relation between the semiotic levels is far more complicated than the diagram shows; insofar as in dynamic situations there is an interdependence between them, the borderline becomes fuzzy. This semiotic distinction, successfully applied in formal logic and then taken over in computer science terminology, is useful in defining structures. Once such structures are designed (the desktop metaphor, to be discussed later on, is an example), we have to modify/amend

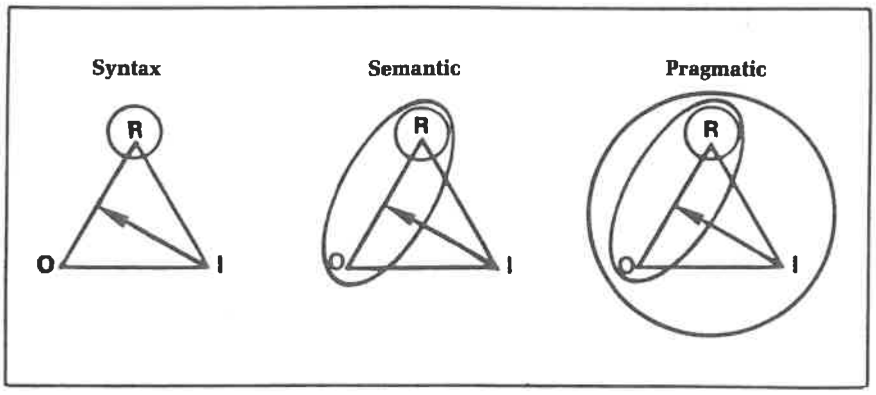


Figure 3. Semiotic levels.

them by considering the dynamics of sign processes. No sign can be considered independently of its relation(s) to other signs, be these similar (such as words in a given language) or different (words, images, sensory perceptions, etc.). The interdisciplinarity of semiotics is a consequence of the fact that sign processes are heterogenous by their condition, and that, in order to understand how different kinds of signs constitute interpretable strings or configurations, we have to acquaint ourselves with each different kind, as well as with the principles governing human or machine interpretation of such strings or configurations.

DESIGN AND SEMIOTICS

Design comes about in an environment traditionally called *culture* (currently identified as *artificial* through a romantic distinction between natural and artificial) and acts as a bridge between scientific and humanistic praxis. Along this line of thinking, Herbert Simon (1982, p. xi) stated, "Engineering, medicine, business, architecture, and painting are concerned not with how things are but how things might be—in short, with design." There is no human activity that does not contain some design component. Quite a few attempts are made to put together a unified design theory. The concept is very broad, while the forms embodying design are becoming narrower and narrower, i.e., more specialized. This is also the case with interface design. But no matter how specialized design becomes, there are certain common elements which make products remote in function share design qualities socially identified as appropriate, good, useful, aesthetic, efficient, etc. Due to the role of computers in supporting and improving human activity, it becomes crucial to identify means for improving design quality, and eventually to develop computational design. Automatic programming is certainly one step in this direction. Progress in the design of interface will follow once we better understand the nature of design and the specific elements of human-computer interaction.

One way to achieve this goal is to look at design principles as semiotic principles—in other words, to consider design as an activity through which designers structure systems of signs in such a way as to make possible the achievement of human goals: communication (as a form of social interaction), engineering (as a form of applied technical rationality), business (as a form of shared efficiency), architecture, art, education, etc. In the case of genetic engineering, the DNA model became the basis for a new field of research and a new industry. The same can be said about computer science, which has as its domain the processing of a precise type of sign—the *symbol*—according to the rules of Boolean logic. Programming languages are *designed* languages; programs are designed too. Computers, with their designed architecture and operating systems which specify the sequence and structure of data processing, reflect our acquired knowledge of the human beings who conceive and produce them. Haber and Myers (1982, p. 23) noted, "Human beings have their own architectures and operating systems that determine how they process information." The organizational principles of the human being as sign producer and sign interpreter, and the organizational principles used in designing computers, should be carefully considered when we design the entity/ies through which users will interact with computers. Everything we know from semiotics about how people devise, interpret, and use signs can and should be used in the design of entities through which interaction will take place. As already stated, this interaction takes place not at the level of physical (or biological or chemical) reality, but through the intermediary of sign sequences (letters, words, icons, programs, interrupt messages, calls of routines, etc.). With our new tools, we no longer interact directly—as with the hammer, the screwdriver, or the plough—with whatever we wanted to change or use—but indirectly, using the mediating function of commands, programs, and expert systems. Accordingly, the breakdown of a tool is primarily a conceptual problem (in a given social context) and not one of physical relevance.

INTERFACE DESIGN: SCIENCE OR ART?

While it is true that the interaction between humans and tools became fashionable in the "computer age," the concept of interface is actually a product of human culture (seen as an artifact environment). It is in this respect that Simon (1982, chapter 1) regarded "the artifact as interface" and "the environment as mold." Design of interface is not just concerned with users and computers, but also with human-to-human relations, especially in the context in which our relation to tools, human contact, and interinfluence become more and more indirect, mediated through entities to a large extent programmed. Signs fulfill the function of *intermediary, go-between,* and *medium.* Some signs can be very precise: mathematical formalism, instructions for booting/rebooting a system, E-mail addresses, passwords. Others are less critically defined: contents of log-in/log-out files in time-sharing systems, names of passed parameters in complex programs, E-mail paths, the place a document occupies on a multiwindowed screen. The designer of an interactive editor has to look in both directions: to the architecture and operating system of the computer and the "architecture" and "operating system" of the user. The two are rather different. While, at the computer end, the designer will assume uniformity and homogeneity (at least for a line of products, and until the succession of improved versions starts), at the user's end there is no such thing as a *general* or *abstract* user. This makes the designer's task quite complex and explains why the computer community is so frequently frustrated.

Schneider and Thomas (1983, pp. 252–253) pose two questions that express the feeling of that community: (a) Why isn't the design of computer interfaces more like science? (b) Why can't the people who design interfaces be more like engineers? These concerns stem from the recognition of the role interface design plays in the human use of computers. In the past (of timesharing and expensive hardware), interface, although important, was less critical. With computer technology becoming cheaper and expanding to more segments of society (especially in the personal computer environment), and in view of the diversity of utilizations of this quasi-universal tool, interface design issues (together with networking) acquire more importance.

If a science of interface design for computers or any other kind of machine is conceivable, then this science will have to integrate in its body of knowledge semiotic concepts and experimental data regarding the human use of signs. On the other hand, computer scientists and engineers should have no problem in understanding the nature of interface design as science *and* art; as information engineering and design; as communication and expression. Despite the diversity of signs and sign processes used in interfaces, these all fulfill the basic function of intermediary between users and computers. Their meaning is determined by what we actually do with computers. The contingency of each mediation—its *likelihood, relative predictability,* its *dependency* on and *conditioning* by other factors—that is, the contingent nature of each interface, is a reflex of design's dual nature as science (in respect to scientific principles of design applied to computers) and art (in respect to a particular, original way of designing). In suggesting attention to semiotics, I do not intend to reduce everything to semiotics, but to point to underlying processes taken for granted for a long time, but not yet integrated in a methodology for improved design interface. The fact that there is more than one answer to the question of efficient interface with a system seems to worry those inclined to accept only one answer to an engineering problem. Such people forget that programming, while a very rigorous activity, allows for the use of creative algorithms and their creative interpretation. Two programs for the same application can be as original and innovative as their authors. The scientific nature of logic reflected in the scientific nature of the computer, as well as the scientific foundation of semiotics, implies the art of reasoning and allows for an *art of computing* expressed in elegant, balanced, optimized codes. The late recognition of the fact that computers are basically sign processing devices (see "the concept of symbolic computation"), and not merely number crunchers, was not just a matter of terminology. It made possible progress in our design of computers and implementation of understanding (intelligence) algorithms.

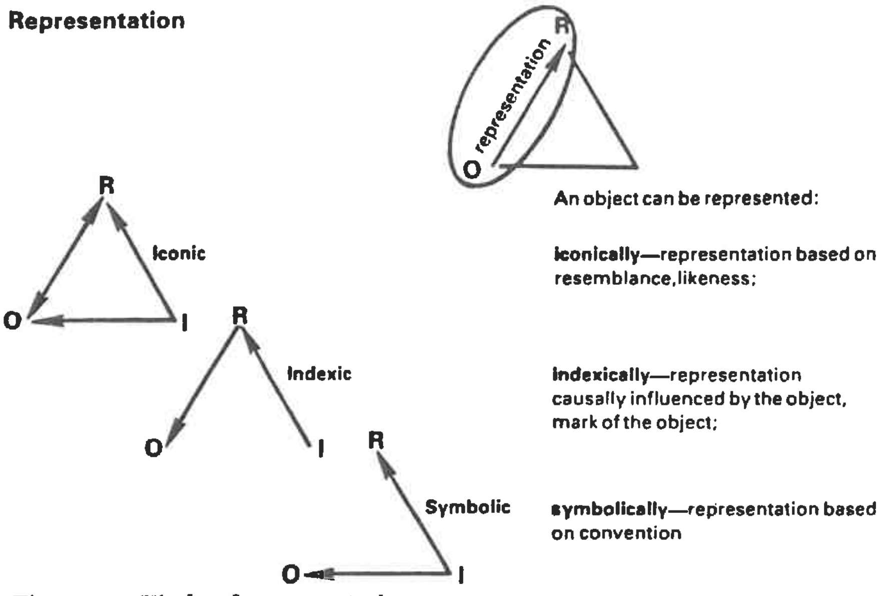
All that we understand or know, we know through the intermediary of signs and *in* signs. (Gibsonians of all shades and nuances will of course disagree.) And all that we apply from our knowledge is semiotic in nature. This observation may seem banal. But for a society still trying hard to explain that there is nothing in a computer which was not previously put there by those who built it, wrote programs for it, or stored data in it, it makes sense to clarify where knowledge comes from and how it comes.

REPRESENTATION

At the beginning of the computer age, which means not only the abacus, but also Babbage's analytic engine or Peirce's logic machine, representation was kept very close to the Boolean logic which was supposed to operate on the represented items. In truth, black and white feats or similar simplified representations (in two-valued logic) of the basic repertory of signs called "yes" and "no," or "one" and "zero," sufficed. More was difficult to handle. Leibniz noticed so much ahead of his time that a language of zeros and ones, while very handy in view of its reduced repertory, was complicated, since it entailed very complicated sign operations applied to the repertory. He hoped that semiotics (which he anticipated) would come up with precise rules for generating not only the equivalent of sentences from natural language, but also mathematical formulae, music, as well as universal ethical principles.

Today we use several layers (assemblers, compilers, interpreters) and operating systems to coordinate the flux of activities that users require as an intermediary between zeros and ones of machine language and high level human reasoning involving the use of language and other sign systems. In order to provide the user with efficient means for reaching his goals in using a system, we try to make it transparent to the user. Interface helps in this endeavor. The specifications according to which they are designed determine the degree of transparency assigned to particular groups of users. Representation of objects, action, data, process status, etc. is a matter of the relation between the representamen and what is represented (the generic object in the sign definition from Figure 1). Representation, and the consequent interpretation of this representation, can take three forms (see Figure 4).

Let us discuss the implications of choosing one of these fundamental modes of representation. Based on a relation of resemblance between the object represented and the iconic representation, the use of icons implies recognition, i.e., a shared environment. Users familiar with the object represented have no difficulty in associating the icon with some object they have used before. However, if they have no prior experience with whatever is represented, if the experience is different from the one intended, or if the convention of resemblance is not carried through uniformly, interpretation is affected. When the *Metaphor for User Interface Design* was introduced at the Xerox Palo Alto Research Center, the intention was "to make computer systems accessible to many people and to make computer



An object can be represented:

iconically—representation based on resemblance, likeness;

indexically—representation causally influenced by the object, mark of the object;

symbolically—representation based on convention

Figure 4. Kinds of representation.

systems do many more things for people" (cf. Goldberg & Robson, 1979, p. 157). The first goal demanded simplicity, and was later achieved, to a certain degree, by using the simplest form of representation, i.e., iconic. The second required complexity (of available programs first of all). In order to find the optimal compromise, designers of the system opted for filtering templates for implementing software interfaces between various groups of users and a complex computing system. Filtering templates make possible the display of graphic representations. They are invisible layers which support the iconic interface which was afterwards, with direct participation of semiotics (the concept of icon), implemented in Xerox's Star Workstation (cf. L. Tesler, personal communication, Cupertino, December, 1983).

Among the indexical representations used in interface are account number/name, passwords, and memory management (how the user learns about available memory or memory limitations). In the UNIX operating system, there are quite a few indexical signs. In the attempt to implement an iconic version of UNIX, designers noticed the difficulty in dealing with such representations. However, indexical signs proved very important for identifying "break-ins." The entire computer security issue brought the topic of indexical signs into the design of interfaces, supposed to be friendly to the legitimate user and impenetrable to information thieves. The symbolic level of representation makes use of high level conventions, such as those of natural language and of logical or mathematical formalism.

The need to establish a coherent level of representation and to avoid changing the convention emerges as a precise normative requirement to be observed when interfaces are designed and evaluated. What this means in concrete terms will be explained when examples are given of actual interfaces and work done.

CONCEPTUAL AND PERSONAL MODEL

Based on these elements, I shall introduce a generalized concept of interface and then apply it to a computer system on which I worked as consultant for the Apple Computer Corporation. First, I should point out that interface, no matter what kind, specifies the optimal set of signs for the interaction between two entities, be they animate or inanimate. In a limited sense, user interface specifies the action the user is supposed to take in order to access different parts of a system according to the design of the conceptual model which is the basis of that particular system (see Figure 5).

In the case of computers, Meyrowitz and van Dam (1982, p. 323) ascertained that user interface, together with the conceptual model, constitute the interactive editor. According to this conception, user interface contains the input devices, the output devices, and the interaction language. In what follows, this view will be contradicted, since I consider the interactive editor itself an interface. Moreover, *every* point of contact between the computer and the user will be integrated in the extended model of user interface, from product design to service (support, documentation, tutorials, seminars, packaging, etc.). What makes things a bit more complicated in comparison to the most common social forms of interfacing through the intermediary of natural language is that *user interface is part of the computer system.* As we know, it participates in, and sometimes supports, process interfacing among different components of the system. Top level interaction with the user through the formal programming language also falls in the sphere of user interface activity.

In the spirit of accepted language theory (in particular, generative grammars), language can be understood as a generative mechanism given as a grammar to be applied to a vocabulary (or, more generally, a repertory) and rules to generate valid expressions. The personal user model, reflecting the way users perceive interface under given operating conditions, is an interpretation of the interface sequence of operations. It is formed in a long process which sometimes starts with tutorials, at other times with reading instruction manuals, and not infrequently with simply trying commands or operations based

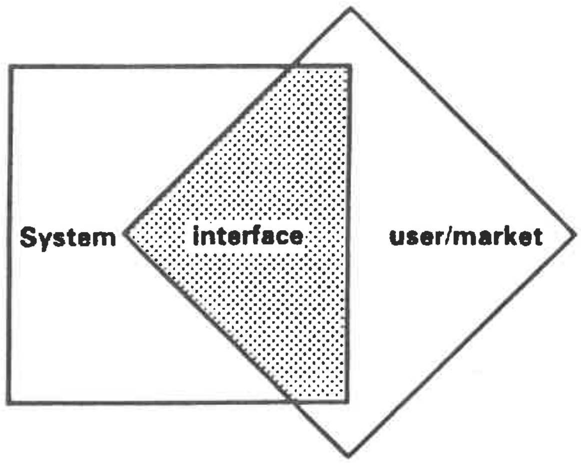


Figure 5. Constitutive elements of the interface sign.

on prior experience while looking for some online documentation (help facility, error messages, or menus, in the case of menu driven machines). In each of these cases, the user tends to extend what he or she already knows. In doing so, his or her semiotic competence suggests two assumptions:

1. uniformity of rules
2. consistency of the representation convention adopted.

These two assumptions are actually semiotic principles derived from empirical observations regarding the way people with a certain competence act when faced with new sign systems or with new languages. Although operationally different, the conceptual model—obviously anticipating the user model—has to integrate these two requirements. In order to give an idea of how semiotic methodology can be applied, I shall concentrate on a common example: the so-called office system computer. The premise for considering a computer's interface from a semiotic viewpoint is that it represents a *complex sign system;* specifically, it represents a system we interpret as an emulation of *the office.*

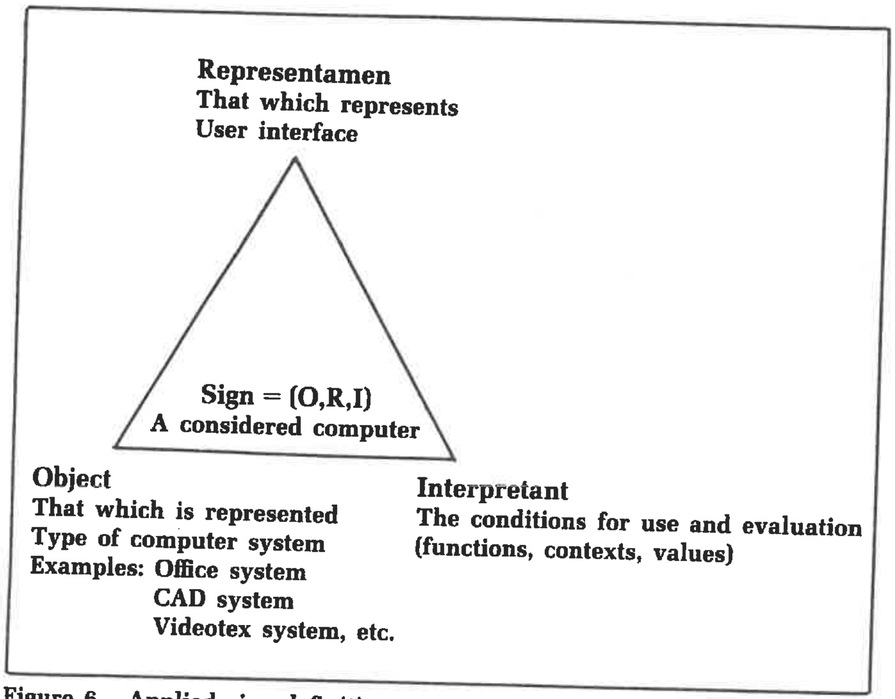


Figure 6. Applied sign definition to computer.

Figure 6 helps in defining the elements of the basic semiotic entity, i.e., the sign, as constituted in the computer context. I have already mentioned that user interface makes transparent in three ways (iconic, indexic, symbolic representation) whatever can be accomplished with the help of computers. The syntax of user interface, although usually influenced by the semantics (the applications available), is independent of the program's syntax. This is not the case with the applications, although the user does not necessarily know this. Sometimes we notice that interface designers impose syntactic rules which make more sense to programmers than to users. Dennis Wixon, in his helpful comments to a previous version of this chapter, presented the opposite position, according to which the issue is to "decrease the distance between it [computer] and its users, much as the hammer is an extension of the arm." If computers were only such extensions, the use of semiotic concepts would not be justified. In fact, they are extensions of our minds, for which reason we have to avoid imposing on the user conventions as rigid as those required in programming. The interpreter should be aware of conditions for use and evaluation as dictated by the activity to be performed (word or data processing, planning, designing, communicating, etc.) and not by the applications available. In other words, users should have enough freedom to do what has to be done in a way which will not make them servants of the machine. If, in the process of constituting the unity *interface–applications–conditions for use and evaluation* represented by the diagram in Figure 6, one or the other of the elements imposes requirements affecting their reciprocal relation (e.g., interfaces not transparent enough for the applications they represent, applications difficult to use or inappropriate to the desired activity), the need to improve the general design of the machine should be recognized. Videotex applications provide an example of this situation. Despite all the investment made in the United States, videotex did not succeed because, among other reasons, the user interface did not adequately represent the applications (electronic banking, shopping, mail, travel schedules, etc.) and the conditions for use (via telephone lines) made it cumbersome.

Some examples of well-constituted semiotic entities nevertheless prevail in the computer market. Without going into detail, I would like to mention office systems offered by Xerox and Wang, workstations produced by SUN and Apollo, graphics machines from Raster Technologies and Silicon Graphics. In these and other cases, attention was given not only to technical performance (speed, memory), but also to how this desired performance is made available to users working under different conditions and needing to accomplish work often not anticipated by system and interface designers, or not entirely understood by them.

LISA—AN ABANDONED OFFICE SYSTEM

The constitutive elements of the office system for which I did consulting work (together with graphic designer Thomas Ockerse) were the desktop metaphor (representamen using iconic representation of applications appropriate to an office system), conditions for use as defined by office work requirements in the context of improved productivity, augmented communication, and integrated activity. The *pragmatics* results not only from the functions made available (word processing, ledger, listing, etc.), but also from the recognized need to integrate them. Everything used in this representation of the office constitutes part of a *repertory*, while the rules of usage, as applied in the process of interfacing, define the grammar of the interface language. Once the product becomes available, the final result that the designer and user look for is not the value of true or false, as in formal logic, but *meaning*.

This requires one more definition of semiotics: Semiotics is the logic of meaning. As such, it approaches the logical laws of sign processes conveying a certain meaning to an intended interpretant (i.e., the *process* of interpretation in which various users become involved, the use of the system). In order to design the interface (representamen = that which represents), the *low level protocol* has to be established. We came to the conclusion that an office is the unity of the *environment, tools, supplies,* and *activities* which make possible the pragmatics defining the specifics of each particular office. There is no such thing as a universal office. There are different types of offices, and when a computer is identified as an office system (IBM's, DEC's, Wang's), this identification opens the door to interpretation and different uses. The list to follow, a low level protocol description, presents an office as our society considers one to be. Once a decision for specialization (insurance, financial planning, law practice, medical, etc.) is made, under the assumption that the production of a specialized computer or the implementation of a specialized software package is justified, the description becomes more specific. In other contexts (the European market, Far Eastern office activity, etc.), the low level protocol will look slightly different (see Figure 7).

The semiotic condition of metaphors is that of expressive ab-

ENVIRONMENT	ACTIVITIES	TOOLS	SUPPLIES
physical space	typing	furniture	pencils
– architectural	editing	– desk(s)	– black
– interior	dictating	– chair(s)	– colored
space filled w.	formatting	– shelving	pens/markers
objects (rugs,	accounting	– storage cab.	– black
furniture, plants,	payrolling	– supply cab.	– gray
pictures, tools . . .)	– employees	– file cabinets	– colors
lighting	– sales	– safe	erasers
– natural	– expenses	machines	liquid paper
– artificial	calculating	– copy	glue
human interactivity	financial modeling	– dictation	paper
– w. personnel	cutting/pasting	– shredding	– plain
– w. clients	representing	– binding	– graph
communication	planning	– typewriter	– colored
environment	tracking	– calculator	– tracing
controlled environment	– inventory	– paper cutter	carbon paper
– w. specified areas	– schedules	stapler	acetate sheets
– w. rules inside	analyzing	scissors	spread/ledger
outside	preparing tasks	letter opener	sheets
– w. rules for	developing tasks	rulers	stationery
legal entity	meeting	straight edge	invoice forms
public environ-	presenting	protractor	billing forms
ment	serving	desk lamps	memo pads
	informing	clock	telephone
	communicating	telephone	pads
	answering questions	trays	labels.
	telephoning	– in/out	stickers
	advising	– letters	file folders
	controlling (quality)	waste basket	binders/cases
	filing	tape dispenser	tape cassettes
	retrieving	file organizer	rubber bands
	listing	desk organizer	clips/paper
	reporting	clipboard	clips
	centralizing	rolodex (open card	tapes (clear/
	keeping records	files)	masking)
	inventorizing	business card file	string
	recording	hole puncher	
	performing	light table	
	cheating	magnifying glass	
	hiding	postage scale	
	working in private	desk pad/blotter	
	pretending	calendar	
	entertaining	telephone books	
	living	dictionaries	
		secretary's	
		handbook	
		thesaurus	
		copy stand	
		numbering/date	
		stamp	
		rubber stamps	
		stamp pads/stamp	
		ink	
		decorative-pictures	
		objects	
		– plants	
		– rugs	

Figure 7. The office environment.

stractions. They are generated, for poetry or science, by way of selecting characteristics according to an intention which the metaphor makes explicit. A desktop as such is not a metaphor, but a working surface. The desktop becomes a metaphor once it is selected to represent the office environment. Behind the desktop metaphor, there is actually another layer which implements a more general model, the layer of windows, a filtering template used for displaying graphic representations (text or pictograms, diagrams, etc.). Windows are components of the repertory and can vary in size. The possibility of arbitrarily overlapping windows facilitates the analogy of paper on an office desk. Obviously, not every system is designed to support covered windows (tiled window managers do not); i.e., the desktop metaphor, as any other metaphor, requires that the designer of an interface which implements desktop conventions use a system that can support it well. Raster display is a necessary condition.

The Lisa machine, which Apple developed using or adapting previous results obtained by Xerox, was supposed to fulfill the goals of user friendliness ("accessible to many people") and complexity ("do more things for people") by providing a very transparent user interface and an integrated, multitasking computing environment. The advantages of implementing Alan Kay's covered window model (accommodating several large displays which could not fit together side by side, making better use of screen, maintaining familiarity with the convention of overlapping paper on a desk) were to be augmented by a mode-free environment. Since a graphics interface was considered, the list in Figure 7 was soon turned into a visual representation of the office.

SYNTAX OF INTERACTION

After defining interface as representamen and specifying its elements with the aid of low level protocol description, and after defining the type of representation (iconic, indexical, or symbolic), we have to relate these to the computing environment, to the integration language. Larry Tesler (1981) explained what this means from the perspective of the user we considered: "How do I do this?" "How do I get out of this mode?" as opposed to seeing on the screen what is available, choosing with a pointing device (which moves a cursor wherever the cursor is needed), and switching from one application to another. Since this issue is critical for interface design, let me explain it briefly and give an example.

User interface consists of input and output devices and interaction language. The latter consists of a repertory (constitutive elements which can be put together to form meaningful commands) and the equivalent of a grammar (rules to be applied). We can input character strings, commands, and coordinates. We can receive the same as output, as well as formatted text or data (or both). The syntax of interaction languages can be prefix, postfix, or infix. Once we adopt a conceptual model for the user interface, we implicitly adopt an adequate syntax for the interaction language. The two aspects are interrelated. Until the Palo Alto Research Center unveiled its then original iconic interface (the Alto [R] station) and the Smalltalk environment (Thacker, McCraight, Lampson, Spronll, & Boggs 1979; Tesler, 1981), the main type of command was the prefix. Basically, a prefix syntax command specifies first the verb (operation) and then the object of the operation (example: edit; file to be edited). The postfix syntax command does just the opposite, allowing first for the selection of the object (file) and then for the desired operation (append). It requires a subject-oriented interface language; i.e., it makes possible a visual language, i.e., a graphics interface. The infix command implies the existence of several operands, each action being virtually connected to such operands.

Let us consider the way the world of typography is emulated on computer-driven typesetters as an example (the Mergenthaler OMNITECH 2000 to be precise) that points to the inherent constraints of a prefix system (see Figure 8).

The pragmatics of the emulated world, on which the culture of the printed medium relies quite extensively, determined the functions of the new machine and consequently its interface: What used to be the composing stick was turned into a sequence defining the font (■T#sp), size (■S##sp), lead (■L#sp), and measure ■M##sp (in picas and points). The text to be typeset immediately follows ■M##sp, the highlighted specifications; hardware (the stick) becomes a software specification. Pagination requires a formatting command, quite opaque to the user (defining page area, zero position, negative area, etc.). Obviously, the change from the technology of linotype (hot type) and letterpress to the technology and interface of the digital typesetter is impressive. Desktop publishing, which was supposed to become a Lisa function, shows not only how technology has changed, but also how changes in interface (from the abovementioned awkward prefix commands on the Mergenthaler to the more direct specification of today's systems) facilitated access to more complex computing for more people.

[R] Registered trademark, Xerox, Inc.

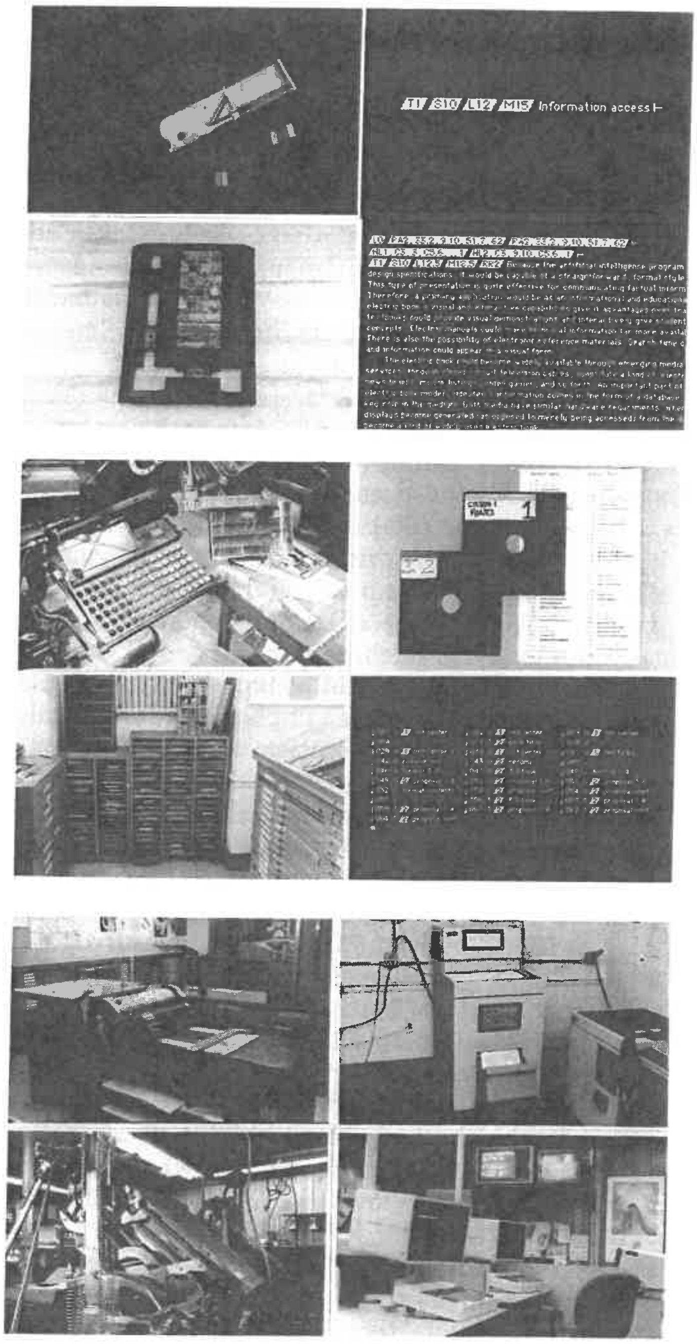


Figure 8. Computer emulation of typographic environment.

The infix command implies the existence of several operands that are established through user interface design. Since the sequentiality of computer string processing is a structural given—and natural language is sequential—the first, and still most common, design interfaces use the sequential paradigm. The relation between the user and the computer is established during the typing in of strings (text or abbreviations) for names (commands) and for operands, sometimes as close to natural language as possible, at other times according to other conventions. Intuitively following the semiotic principle of feedback, the designers of such interfaces made sure that the commands are echoed on the output device (screen, in this case) before or after being processed by the editor. The interface design becomes more and more critical when online activities are performed. The old, hot metal typesetter was an online device. Each key command ended up in the mechanical selection of the appropriate mold; the hot metal was poured in the mold and the result was a line. Norpak's IPS-2 system (which supports a Telidon alphageometric system) duplicates the real world of design and typography, moving beyond the infix structure (to a mixed environment in which infix and postfix commands coexist, with a drawing tablet that allows for the actions connected to some of the available operands). Interface preserved some basic tools: compass, pantograph, ruler, triangle. But the end products (image on an editing screen as opposed to newspapers at a delivery station) are obviously different. There is also a limit to the emulation, clearly illustrated in the way people interface to a newspaper as opposed to the selection pad of a teletext-videotex-television device.

Up to this point, we have examined different ways a strategy of imitation is pursued and interface designed to make the imitation clear to the user. The limitation imposed by the postfix syntax on the interface adopted for Lisa can be partially overcome by the introduction of function keys, the use of macros, better emulation of natural language, design of "intelligent" editors, etc. The postfix command opened the way for multiple choice of strings and/or visual images (graphics interface which is mainly iconic). But, to date, less has been made known about the semiotic characteristics and limitations of such interfaces. In the process of semiotic evaluation of the postfix syntax and its use, conceptual limitations as well as experimental findings were considered. Adopting the postfix syntax was only one more step in the direction of better understanding the perspectives for improving interface over the course of time. For instance, frequently the basic distinction object–action (reflecting the noun–verb distinction in natural language) is not clear–cut. People also tend to transform nouns into verbs (and vice versa). Moreover, and for reasons implicit in the semiotic structure of such interface, selection and response times sometimes increase over the time of operation itself (opening or closing a file is the most obvious example). In what follows, particular aspects of the office system interface design will be more closely considered.

The following model was used in evaluating the interface specifications (see Figure 9):

Since the desktop metaphor and the iconic representation were chosen for the object (of action), the specification of the action had to use a graphic form, too, but a different type. The user should not be put in the confusing situation of an undifferentiated representation of objects and actions. Because actions are difficult to represent in static pictograms (they can be represented in animated sequences), representations of actions were not iconic, but symbolic (through words). These words are displayed in pop-up menus and organized according to main categories corresponding to the state of the machine (after booting, once the application was started, before and after editing, etc.). Since window sizes can be changed and both characters and full bitmap graphic operations are supported, the interface had to be designed so as to preserve the realism of the convention. Once the window size is changed, the user sees only a portion of the document and not a sized–up/down image.

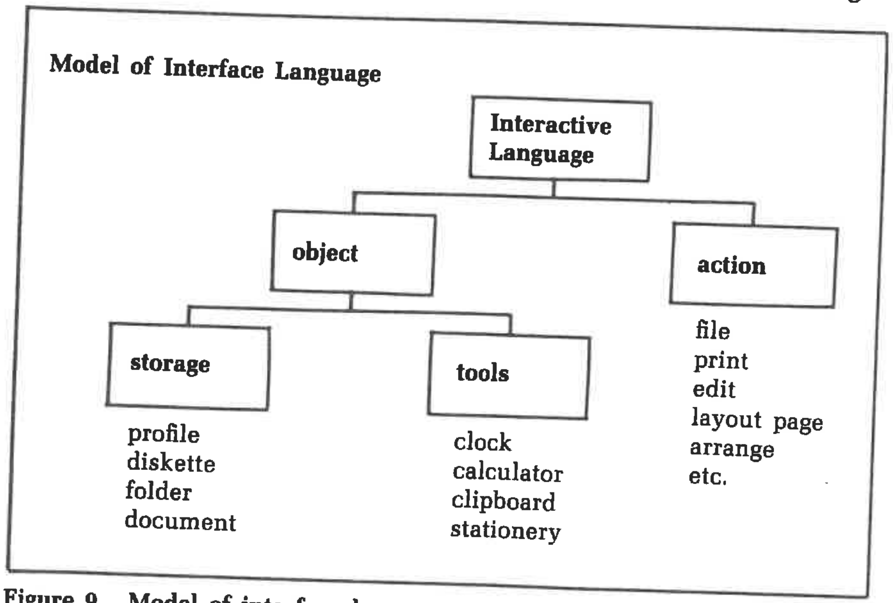


Figure 9. Model of interface language.

Interface design requires *semiotic consistency;* i.e., it becomes a matter of uniformly using whatever means of representation are considered adequate.

Although every effort was made to ensure consistency, later on, during an evaluation in which design principles were applied to see how successful the designer had been, it became evident that the clear-cut conceptual distinction between actions and objects was really quite fuzzy. Instead of documenting the successful parts—in the meantime acknowledged by their widespread use in interfaces designed for other systems—I prefer to discuss the critical aspects and what means semiotics provides for evaluation and redesign.

The type of representation chosen (iconic, indexical, or symbolic) influences the user's interpretation. The following representation of the Lisa calculator visualizes this idea (see Figure 10).

Recognizing that the squares on an icon called *calculator* are "real" buttons which have to be "pressed" (not on the screen, but

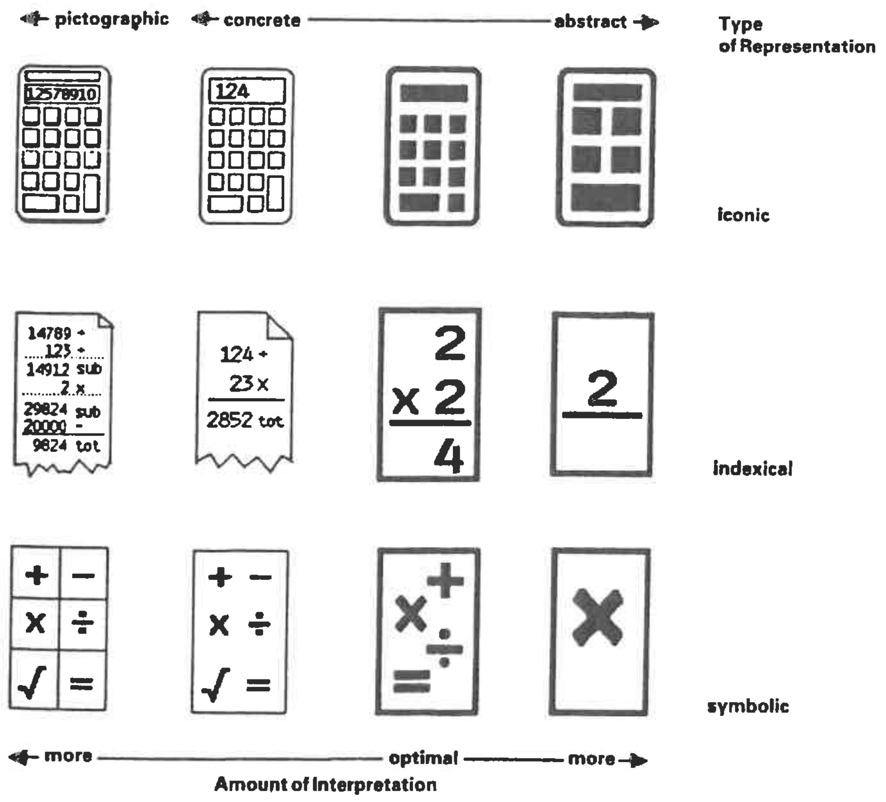


Figure 10. The relation between representation and interpretation.

via the mouse button) involves an interpretation different from what the pictogram of a stack of paper suggests—and again, different from the convention of a clipboard which holds the last item cut or copied, but which cannot be edited. The LED-type of display and the numbers displayed are a kind of convention-over-convention. The user is confronted with a real calculator and a representation (iconic) of a pocket calculator. This is a difficult semantic situation, similar to the one we face if some of our words were at the same time the objects they denote! Figure 10 presents the visual analysis a designer should go through to evaluate options that might prove better than the initial one.

The easiest and most direct interpretation is the one resulting from information sufficient to facilitate recognition. After semiotic inconsistency, the second issue is whether icons should be presented with the names of the items they represent. Recognition memory is better for iconic representation. However, the recognition of tasks, such as how to use the calculator, seems to imply that dual coding (images and words) is sometimes confusing, because users will typically wonder why they need a label underneath an image they recognize. Is there something they might not understand?

Computers such as Xerox Star, Intran, and later Macintosh added the interactive editor formatter to the postfix command that is part of the interface language: "What you see is what you get." While this is a welcome quality, soon to be adopted by other computer designers, it has the drawback which Brian Kernighan identified as "What you see is *all* you've got." In the opinion of several interface designers, it is uninformative and gives no clues to what influences a certain format, why some changes are not possible, and why there is no consistency between formatting capabilities or between the different applications used. The filtering templates used in such an interactive editor formatter are as important as the input messages in the prefix commands. On most available systems, the semantics of the templates is confusing and not consistent with the visual representation of the objects, locations, or the pop-up menus of actions (based on a computing environment such as Smalltalk and supported by a pointing device connected to the cursor or current position independent manager). Another limitation affecting the use of visual language results from the aliasing condition of raster graphics (which supports the window model). It can be compensated for either by increasing the density of pixels (which results in the need for computer memory and increased response time) or by using multiple bits per pixel (grey-scale displays). This is not only a hardware issue. The higher the quality of images, the better the possibilities to generate a visual language for the interface and to support high quality applications.

HUMAN-COMPUTER COMMUNICATION

An office is not a collection of files, typewriters, calculators, etc. That is, it is not a collection of hardware but an *environment* where communication (exchange of documents, for example, storage and retrieval of data, planning, etc.) is necessary, and interconnection should be possible. This explains why office systems successful in supporting individual office work but not communication have not made it in the market. As far as the semiotic evaluation of the Lisa is concerned, it failed on the market due to severe communication limitations, among other reasons. Apple was aware of some of the limitations. Produced under market pressure and unveiled before the product was actually finished, from the perspective of technology and interface, Lisa claimed an integrated environment within which communication among applications was possible. From the beginning, this communication was limited and difficult to accomplish. The user learned from the manuals that information could not be switched between all the programs, and that sometimes only one-way communication was possible. This was confusing, to say the least. What was worse was the reality that the system performed below these specifications. When trying to use the clipboard as a communication go-between, the user frequently experienced crashes which destroyed a file. Communication with the outside world was possible through the *LisaTerminal* (sic!) within which editing possibilities were kept to a minimum. The emulation of terminals used for electronic mail (especially DEC) was considered sufficient. Implementation of this emulating function was primitive. Local Area Networking was not provided; consequently, while data was processed electronically, it had to be exchanged through hard copy obtained on a rather slow printer. Against the background of these communication limitations (to which incompatibility with other systems should be added), the price tag of $10,000.00 came as a further shock. Nevertheless, I claim that it was not the price but the communication limitations that undermined the product. Later on, when the price was slashed and communication facilities only slightly improved, Lisa—which did introduce very important computing concepts—did not make a recovery in the market.

At this point, it is useful to suggest that more attention be paid to the semiotics of communication pertinent to computer interface.

There are people who question the subject. "Do people want or need to communicate with computers any more than they want or need to communicate with a typewriter?" is a question Dennis Wixon obliged me to consider. Actually, there is no such thing as man-machine communication; this is a way of speaking, a way of anthropomorphizing machines. Communication is the semiotic activity that brings user and designer together by the intermediary of the language(s) they use. Once the user accepts a language, he or she will apply it according to the rules the designer embedded in the interface, and *their* communication, mediated by a certain machine, will take place. The point is not that the user is at the mercy of the designer. After all, aren't we all at the mercy of our parents, who brought us to life in a language we could not choose? Rather, the designer and user, as parts of a given culture, share established conventions and participate in the establishment of new sign systems, if such become necessary. Programming languages are just such systems, i.e., languages which support precise functional activities and make possible new tools, which are cognitive in nature. The field of human factors in computer systems, "an unruly mixture of theoretical issues and practical problems" (Norman, 1983, p. 1), developed as a result of the difficulties computer scientists and engineers face when considering the relation between the systems they build and their potential users. Psychological concepts were brought into the picture first, and previous observations on the interaction between humans and various tools and machines were applied, not unsuccessfully, to a technology very different from any previous one. What was less considered was the fact that signs and sign processing represent the common underlying principle both of human interaction with computers and of the computer, "a member of an important family of artifacts called symbol systems" (Simon, 1982, p. 28). Since the technology upon and for which we build interfaces changes very rapidly (some that we are aware of, some to be discovered), semiotic principles provide a foundation for a more comprehensive specification of interface design (user and process interfaces), and for the evaluation of interfaces currently in use.

INTERPRETATION OF INTERFACE

We know that signs are constituted of structural components in a limited number of ways. For instance, major structural components of visual signs are shape, contour, color, and texture. If an object is distinct enough, shape alone may be sufficient for recognition (an observation that interface designers using visual representations sometimes apply). I shall exemplify this by referring again to the calculator (see Figure 11).

Pictographic representations are very concrete, almost so concrete that, if the context changes and the user is presented with a different pictogram (let's say one of the solar battery calculator), he or she will have difficulties in "using" the sign. The semiotic level is reached when the conventionality of the sign becomes evident. (Convention here means *as convened, agreed upon,* and accordingly shared in a given social context, in a culture.) Once the convention is recognized, the next step in interpretation is associating sign and function. Only at this moment does the user integrate the component of an interface's repertory in what the designer intends the language to become. Obviously, understanding what an icon represents, as opposed to what it pictures, is essential for designing user interface language. Once this is understood and consistently applied, we can decide upon one type of interface or another, or upon a mixture of representations. Regardless of our choice, what is important is understanding the different sign processes (different "grammars") that characterize the three fundamental types of representation. Words can be used as well; and at the extreme of the symbolic representation, one can add the word *CALCULATOR,* or some abbreviation.

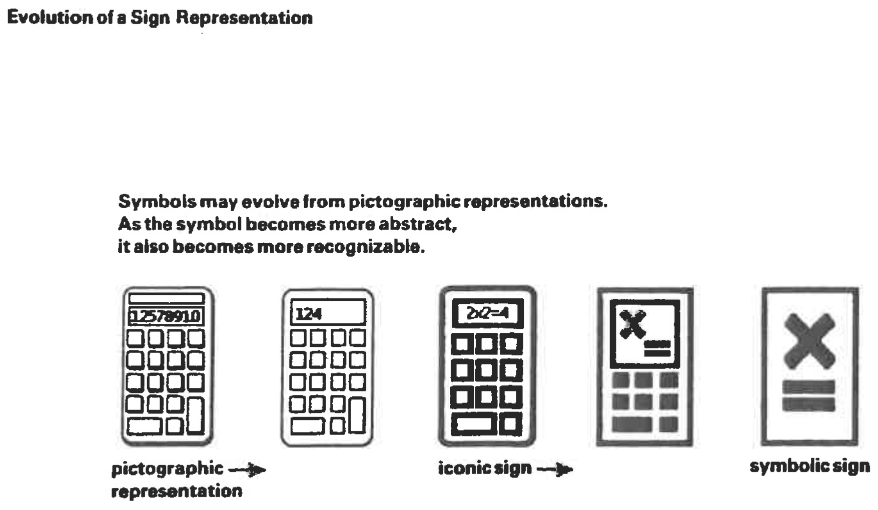

Evolution of a Sign Representation
Symbols may evolve from pictographic representations.
As the symbol becomes more abstract,
it also becomes more recognizable.
12578910
124
2x2=4
pictographic → representation
iconic sign →
symbolic sign

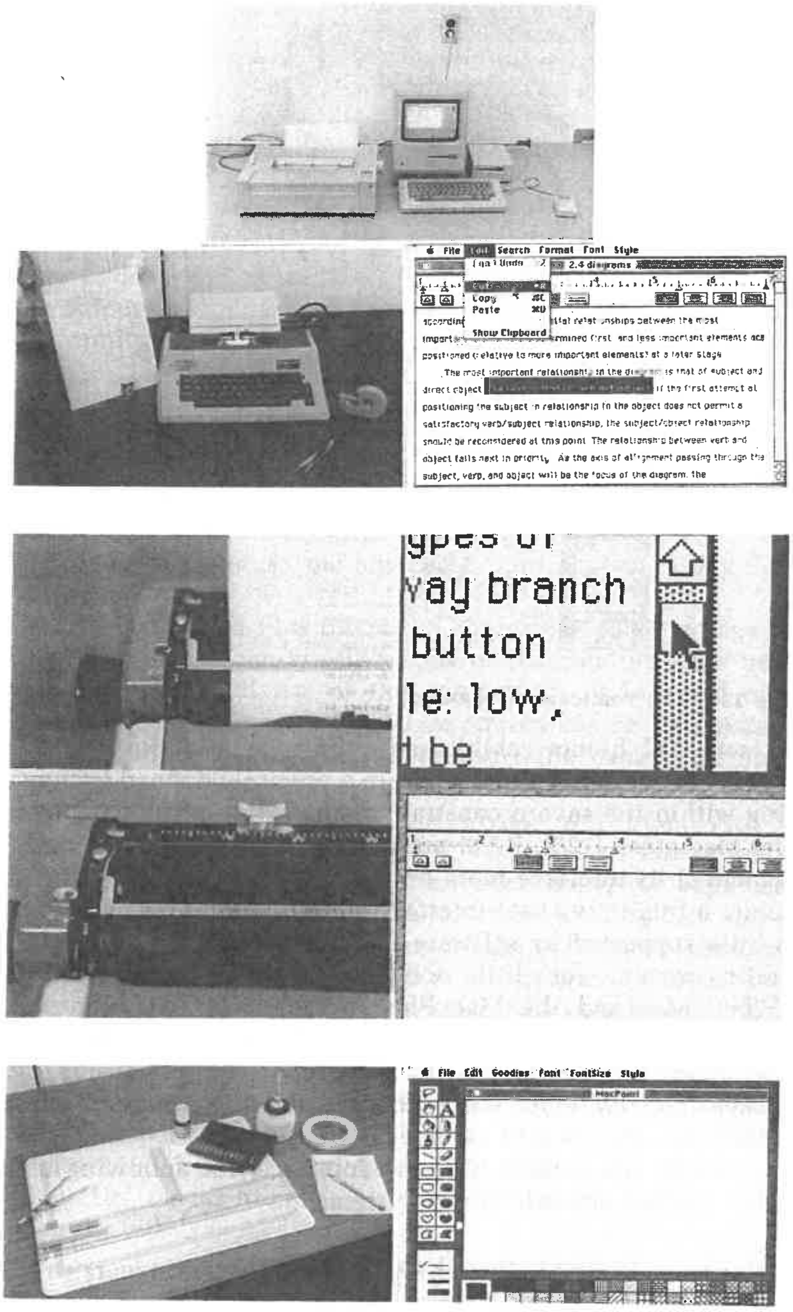
Figure 11. Evolution of a sign representation.

MACINTOSH (OR: REINVENTING LISA IN THE MAC ENVIRONMENT)

The initial step in designing a user interface is to determine the operations and the entities on which operations will be performed. If template filters should be used, the identification must consider object/location as opposed to action/structure. Editing filters are in fact devices that perform the basic semiotic operations (substitution, insertion, omission) according to specifications from the user or the system. The same holds true for viewing filters (used to specify areas of a document to be viewed and to generate viewing buffers). The editing and viewing filters are semiotically equivalent; functionally, they are sometimes identical (screen editors), disjoint, partially overlapping, or properly contained in one another. This is part of the pragmatics of the interface, and necessarily relies on hardware specifications. From a semiotic perspective, which emphasizes the unity between function (interpretation, content, use), syntax, and semantics, there is only one way to proceed in approaching interface: as part of the system, not as a delayed addition to it. Despite its qualities, the Apple IIC (one among several possible examples) shows what happens when an interface concept (the use of the mouse) is adopted primarily for marketing purposes. The same holds true for the Macintosh computer in its initial stage—a product celebrated by quite a few but which, despite its qualities, was considered a hybrid between an office system and a dignified toy. Let us examine some of the reasons for this ambivalent evaluation. As with the phased-out Lisa, emulation was the semiotic strategy used. One might argue that, for the sake of realism, a certain *overdesign* of the interface (emulation of details) makes the Macintosh's very small screen (management of a reduced number of pixels) seem continuously too busy.

Figure 12 exemplifies what kind of representation was adopted: sometimes reminiscent of the tool (typewriter) whose functions the computer can fulfill; at other times, new functions; and finally, icons of pencils, brushes, and paint cans, or of forms already available. More than one convention was implemented here, the quality of the interface consisting of directness but not consistency. What also surprises is the way tools are introduced under the cutesy name of *Goodies*, a mixture of necessary and less necessary elements (see Figure 13).

Moreover, the finder—a remarkable device used in a contradictory way—almost turns into a hidden collection of toys. It is not that computers—especially the PC category—should make users lose



according ... tial relationships between the most
import ... rmined first, and less important elements are
positioned (relative to more important elements) at a later stage.

The most important relationship in the diagram is that of subject and
direct object ... If the first attempt at
positioning the subject in relationship to the object does not permit a
satisfactory verb/subject relationship, the subject/object relationship
should be reconsidered at this point. The relationship between verb and
object falls next in priority. As the axis of alignment passing through the
subject, verb, and object will be the focus of the diagram, the

Figure 12. Macintosh emulation of typing and drawing.

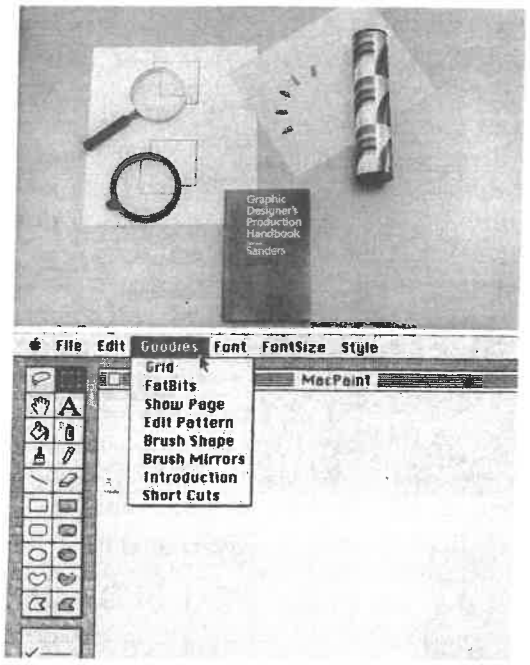


Figure 13. The semiosis of Goodies.

their sense of humor, childishness, or ingenuity. Rather, it is a question of evaluating options within a precisely defined framework. Living within the severe constraints which the development of the initial Macintosh (128K RAM memory) posed should have made the designers of its interface more aware of what is necessary and what becomes a frustrating user interface option. For as long as Macintosh was little supported by software developers, the weaknesses of the operating system were little noticed. Later, this situation changed; the "fat" Mac and the Mac Plus solved some critical problems. Nevertheless, each improvement seemed quite accidental because Apple failed to understand the need for a comprehensive *design language© within which scope, changes and improvements maintain the identity and integrity of the product.*

In view of the semiotics of the interface, the following characteristics proved critical for the system:

1. directness or simplicity (which the Macintosh designers stretched almost to the extreme with visual representations that are sometimes excessive in their details);

© Copyright Mihai Nadin, 1979.

2. consistency or uniformity, observed within applications but *not* among them. The degree of integration is actually less than some believe and Apple advertises. However, a certain modularity is noticeable;
3. permissiveness, unfortunately not clear to the user how far an application goes, and not entirely complemented by what is known as "forgiveness" in the case of mistakes (the UNDO function)—not every operation can be UNDO-ne, but this the user finds out after the fact;
4. operationality, i.e., a set of qualities referring to computing time, uniformity of speed, file management, etc.

Clearly, these characteristics are related. While Macintosh is a very fast machine (assembler programmed), in its initial stage it had poor file management and consequently seemed slow (psychological differential) at times. When the user switched from the mouse functions to the keyboard, this structural shortcoming became even more evident. Some operations, "cut and paste" for instance, do not follow the prefix mode.

The new Macintosh Plus offers the enhanced speed, storage, and expandability which the specifications of its user interface design require for meaningful use of the conventions it is based on. In fact, the evolution is towards what *Lisa offered before the Macintosh* became a product, especially the multitasking capability, plus a much better communication capability, despite the limitations of Appletalk. Even the input and output devices were improved. The MacPlus has a keyboard with numeric keypad, four cursor control keys, and a new port (SCSI, the *scuzzy* interface) for high transmission speed connections of peripherals, which can further expand the complexity of tasks for which users might choose this machine. The long learning process, at the end of which semiotic implications were recognized and actually used in *Macintosh's* improved interface design, does not stop here. There are other companies which use the advice of designers aware of or applying semiotics to their own work. Aaron Marcus and Associates is one such design firm whose methods incorporate semiotic principles. Aaron Marcus (1983, pp. 63–70 and 1983, pp. 103–123) has written about his work, and I encourage the reader to consult the respective articles.

TRANSLATING SEMIOTIC REQUIREMENTS INTO DESIGN

Part of the desktop metaphor, and certain elements of iconic representation convention, have in recent years been assimilated in the design of interfaces for new machines competing for their share of the market (Amiga, Atari ST, and a whole bunch of "generic" computers). I see nothing bad in this trend, provided that, in the process of imitating a successful approach, designers attempt to understand what made this approach necessary. Generally speaking, the design procedure is exactly the reverse of the interpretive process the user goes through when dealing with user interface language. Only after the appropriate functions are determined is it useful to consider how those functions translate in computing content and memory-related issues (i.e., semantics) and, furthermore, how this content will be represented. If the pragmatics of the system leads to the conclusion that visual representations (for example, icons) are justified, design should be considered only in the greater context of the design *language* system. I would insist that interface language be, in principle, formal, which means that the language should function according to a logical structure that the user is familiar with or can easily grasp, and which, while adhering to the spirit of computing logic, should not contradict so-called natural logic (cultural background as the environment of human logic). Of course, voice input devices—a subject impossible to ignore when predictions present this alternative as almost available—do not make this task easier. In the course of using a given interface, the user acquires a progressively higher level of competence (learns the editor), and, accordingly, user performance in operating a system improves (use of short-cuts, for example). In respect to this, a certain influence, quite often overlooked, is exercised by the type of computing environment: stand-alone (becoming more pervasive in the market), distributed, or timesharing. The constraints each type imposes on the design of the interface should also be accounted for when the sign representing the system is constituted, not after everything else has been defined. Many computers, especially stand-alone units, are offered with all kinds of "cosmetic" interface contraptions added under marketing pressure and offered by various OEMs. This quite often affects the user's performance and adds to the confusion already disseminated by the rather chaotic computer market. No interface language is an entity in itself, even if it enters the market with the backing of the largest companies. In one form or another, they all refer to everyday language(s), the so-called natural language, to the language of gestures, of trademarks, etc. In extension of user interface (documentation, tutorials, seminars, support, etc.), this aspect is even more obvious. While the conceptual model of a system is the *premise* for the coherence of interface language, there is actually nothing that guarantees such coherence. Knowing that the user is in fact represented by a divided cognitive structure, in which sequence and configuration (i.e., time-related and space-related perceptions and activities) are not homogenously supported by the brain, we should be able to design interfaces in such a way as not to affect the balance of these two basic cognitive modes.

Interface engineers might find this distinction irrelevant to their work. But experimental data fully support the fact that switching from a sequential mode (operation of keyboards) to a configurational mode (using painting devices, digitizing tablets, etc.) affect the user and his or her performance. In a good interface design theory, the observation that interfaces represent what we know about computers and the humans using them might prove useful. For instance, knowing how people learn proved important at Xerox Park, where the user interface "transparency" of interface is a cognitive quality supporting sign processes within the general framework of the theory of learning. Comprehending a specific system of signs means to identify the structure of that system. The "transparency" of interface is a cognitive quality supporting an emotional component. As long as we have human beings in mind when we talk about the user, emotions cannot be simply discharged as irrelevant. In order to be made more apparent to the user, interface languages should use (a) *concrete* representations of objects and storage, and (b) operation representations that relate *directly* to actions. Concreteness and directness must be expressed as clearly as possible.

Engineers require that we operationalize these terms; otherwise, the requirement of concreteness and directness remain quite obscure. To make concrete representations of objects and storage, one can start with almost photographic images of the respective items. Directness of actions can be achieved by animated sequences. Whether, in the end, the interface will keep the photographic image or simplify it in a realistic rendition is less important. However, the level of concreteness should be uniformly maintained.

The wastebasket on the *Lisa* user interface has an elegant slant lid. No other object represented on the desktop has a similar visual accent. The icon of the computer is called *Preferences.* Actually this is not the representation of an object, but of actions to be

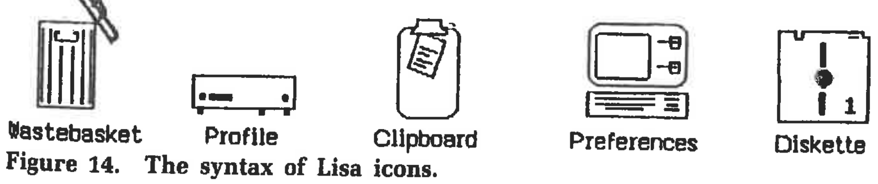


Figure 14. The syntax of Lisa icons.

performed: setting conveniences, selecting defaults, connecting various devices. While, during the design stage, concreteness was not a goal, directness was, supported by the use of the mouse as a pointing device. Evaluation of what is actually implemented allows for improvements. The framework for evaluation is set by identifying requirements—even those difficult to formalize and render operational—which ensure the quality of human–computer interaction. The examination of the syntax of the icons in Figure 14 already suggests ways to operationalize aspects of concreteness and directness.

As already stated, user interface contains the so-called input and output devices and the interaction language as it is developed from/with the conceptual model of a given system, plus extensions (documentation, tutorials, seminars, support, etc.). Obviously, the design of such components (keyboards, tablets, light pens, painting devices, printers, etc.) integrates product design considerations, ergonomy, psychology, marketing, etc., but not always communication requirements. The unity of hardware–software, and their reciprocal influence, is also important. Very few systems available today, as far as I know, were designed to integrate such diverse components. Dealing with the pointing device (mentioned in the previous paragraph as important in achieving directness) as an independent component (there are manufacturers which specialize in "mice") is against the spirit in which this interface component was conceived. There is nothing wrong with specialized manufacture-of—"mice" (or, for that matter, of hard disk units, or other interface components)—if a unifying and integrating design which is faithful to the initial concept serves as a premise. But this is rarely the case. The three-button mouse (e.g., on the Apollo workstation or on VAXSTATION 100) is functionally quite different from the one-button mouse. But it was uniformly integrated in the user interface. The syntax of clicking—once, twice, three times—introduced by Apple affects directness. It is not an extension but a new dimension, complicating the user interface.

USER FRIENDLY: A FUNCTIONAL QUALITY

Ease of use and learning are frequently identified with user friendliness. This is accomplished when the interface is clear, displays conventions familiar to the user, allows the user to infer from the use of one application to another; when the system is reliable and responsive; when help is provided but does not become excessive, and is provided in adequate formats corresponding to the user's level of expertise. Hence, friendliness is not only an interface issue. Unreliable or slow systems are not friendly even if the interface is pleasant. The Xerox 6085 office system is an example at hand. It apparently uses the successful iconic interface of the Xerox Star, while simultaneously imposing upon the user rules of operation (e.g., moving icons, copying, printing, etc.) which, after Lisa and Macintosh, are anachronic. The system is slow. Consequently, the appearance of friendliness and the slick product design turn into teasers. The quality of typography and, for computers with color capabilities, color considerations affect friendliness, too. Rudimentary icons in color, such as the ones used on the Amiga, make clear that without maintaining the quality of design, the iconic interface can degenerate. One critical area—also evident in the Amiga—involves error messages. Errors make the user, even the expert, quite uneasy. The way errors are represented to the user affects performance and the learning curve. Using the appropriate word or image, or their combination, is an important semiotic requirement. Usually, an alert file stores these messages; but it is the routine of the alert manager designed to display them. And this particular routine is usually designed with little understanding of error message semiotics.

Error messages, often anthropomorphized, should be integrated in the user interface language and reflect the degree of integration of the available software. The problem is that, since the software comes from different developers, who receive only system specifications, error messages are improvised without any consideration of the interface requirements. A recent analysis of error messages in the major programs available for the IBM-PC—5 years after its introduction, and with three million machines sold—shows that they constantly deviate from the convention of the user interface. In the case of programs trying to emulate the desktop metaphor in the IBM-PC environment, error messages pertinent to the Macintosh are simply taken over (the icon for a crash, the message referring to the sequence "select object before action," corresponding to the postfix syntax of the interaction language).

An important conclusion resulting from the observations discussed and exemplified above is that, while underlying principles are relatively independent of technology, semiotic principles, as they refer to sign processing, become technologically dependent when applied. This reflects the law according to which the pragmatics of the sign is context sensitive (Nadin, 1981, p. 215). There is no way to avoid the consequences of this law. Efforts in the direction of better programming (sometimes for the sake of programming) or higher technology (sometimes for the sake of technology) are quite impressive. Programming and technology are interwoven. Design weaves them together, and precise design concepts, uniformly applied, ensure the unity of the computer. As mentioned above, interface issues are issues of interpretation (pragmatics) as related to the various types of signs used in interface. Recognition of the object represented is based on two complementary processes:

1. recognizing parts of the object in relation to each other and to the whole, and arriving at some inference based on their interrelationship;
2. recognizing the whole, and inferring from the whole to the parts.

A product's look and functionality are a continuation of user interface, and are related to every other interface of the system. In the course of product design, the formal and the functional qualities should be achieved, while considerations of semiotic unity of the interface are observed. User friendliness refers to physical and mental aspects of working with a computer, programming it, or simply using some of its routines.

Usually, designers know what their computer is supposed to provide; that is, they know what the expected product can be. They also know, or are able to determine, who is going to use the computer (be interested in the product) and what his or her background is. At the system level, the following relations can be established in view of the intended user friendliness (see Figure 15).

Within a given conceptual level, which takes into account the multiple interconditioning of user interface, it is possible to avoid both settling at an arrogant level (if you don't understand it, too bad!) or a primitive form of friendliness (which ends up offending the user). There is no such thing as universal friendliness. This becomes clear at the conceptual level, which shows that each user will form his or her own model when using a given computer (see Figure 16).

Should interfaces entirely specify a course of action (friendliness by holding the user's hand through every step), or provide a free environment in which each user is able to decide his or her own sequence? The question has no clear cut answer. In my recent work on a project entitled *Design Machine,* this issue was brought up by the community of designers for which we want to build a computer able to support not only production, but also conceptual, creative work. Simulation in software of design work showed us that basic

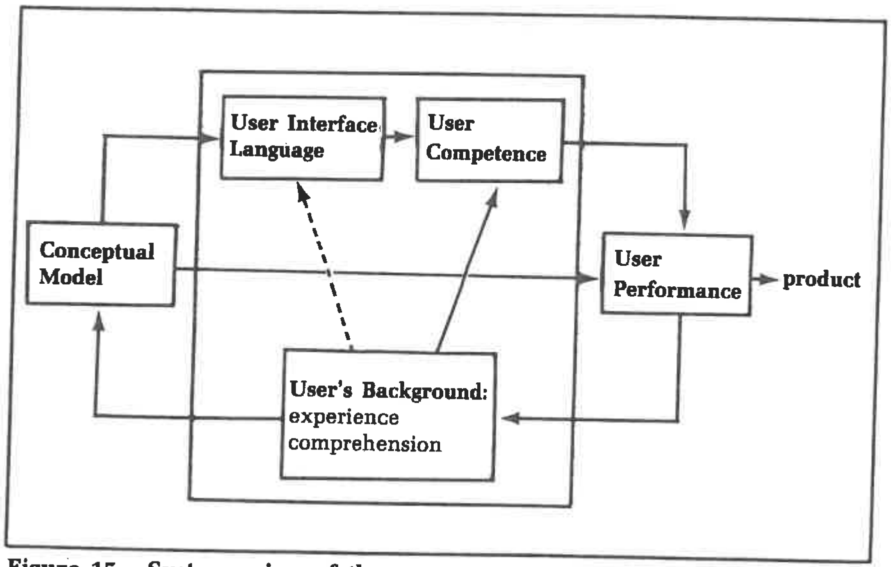


Figure 15. Systems view of the structure of interface elements.

design tools are necessary, but that they have to be conceived as versatile, as "soft" as possible (cf. Nadin, 1986).

The model developed by each particular user (influenced by manuals, guides, tutorials, etc.) is the product of "learning" the system or being "taught" how to use it. Generally speaking, the user employs interface according to the semiotic interpretation given to the interface. This interpretation is based on the user's model. Preconceptions influence this model; so do other semiotic contexts: cognitive skills, emotional factors, aesthetic characteristics, etc.

CATEGORIES OF USERS

The common representation of *the user* distinguishes the novice from the veteran. This is a linear representation, very comfortable, but not necessarily appropriate (see Figure 17). It implies that a novice will sooner or later become an expert, a supposition that is far from confirmed. It also implies that, once initiation is over, the expert must work with the limitations inherent in the system that made it approachable to the novice, i.e., accept the help that later turns into a hindrance. A more complex model is necessary, one that considers the knowledge the user accumulates from working with various computers as an important characteristic of the dynamics of the interface. Although experience is important, a semiotic

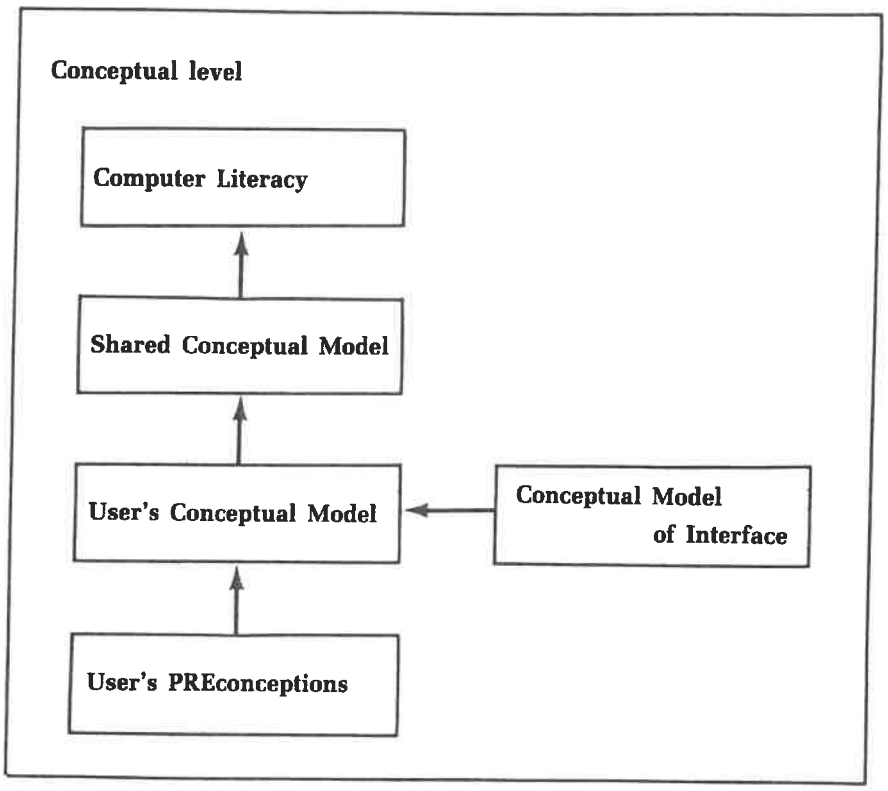


Figure 16. Conceptual level.

Simplified User Model
Naive
Experienced

Figure 17. Simplified user model.

property of computer-aided activity is that, in order to understand and use sign systems, a user has to bring into the activity not only specialized knowledge, but also comprehension skills gained from culture and general education. Computer literacy is only part of this comprehension. The *improved user model* is supposed to help the interface designer evaluate his choice of signs (see Figure 18).

The distinctions *naive, competent, experienced,* and *expert* are not easily quantifiable. Intuitively, designers tend to identify such distinctions through the use of certain idiomatic expressions (words), or by giving up images (deemed primitive). Lisa was designed not just as an office system addressing secretaries, accountants, graphic designers, and other professionals, but also as a computer tool which could support a fair amount of development work. The distinction

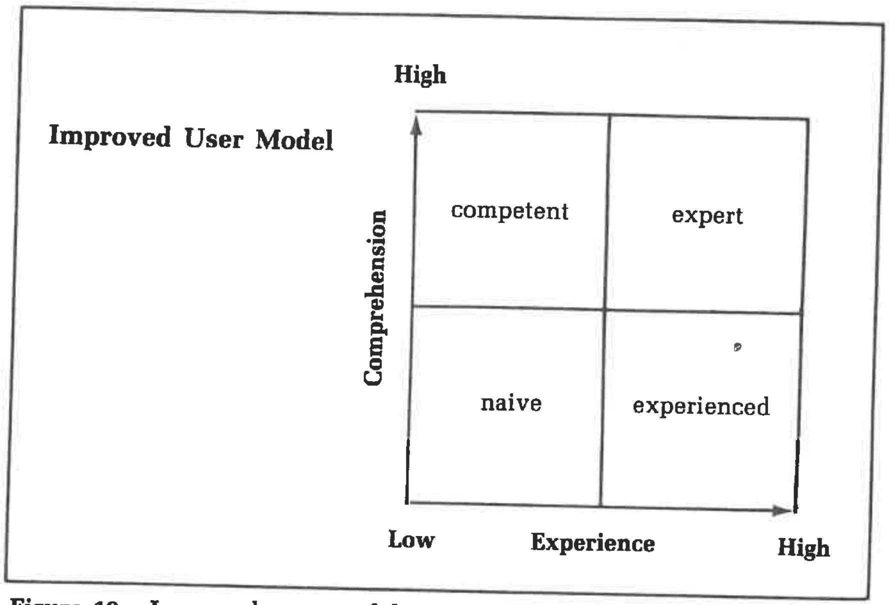


Figure 18. Improved user model.

between the office system environment and the workshop actually reflected the distinction between user and programmer. The desktop metaphor and iconic convention were carried over to the workshop environment only to a very small extent *(Preferences* and *Editor).* Once in the workshop, the programmer faced conventions familiar to professionals acquainted with a programming environment: FILEMgr, SYSTEMMgr, Edit, Run, Debug, Pascal, Basic, Quit? (at the top of the tree). The cursor functions only for Edit and for setting Preferences. A letter confirms each command selection. The tree of successive options goes no deeper than two more layers. The change in the set of signs used (from icons and menus to actions specified in technical terms, from windows to separate process commands, etc.) corresponds to the change of expected user, but in a way in which jargon takes over the interface concept. For some reason, the computer community developed a scientific vocabulary and a stock of almost private expressions—opaque to the nonexpert and humorous (or intended to be so). We know how this semiosis came about. We also know that the energy and inventiveness of young and very young computer developers contributed to a special stratum in our technological culture. In the workshop, friendliness takes the form of options which in the office system would sound threatening—*Kill Process, Scavenging*—or questions which sound worse than what they are—*What?* or the abrasive *Do you really want to. . . .?*

Interface designers could avoid the discrepancy noted here if they considered the users from the perspective of the expected comprehension and assumed experience, as reflected by their professional language (a subset of the natural language they use) and their ability to infer from one environment to the other, from one program to the other. In view of this, the two-dimensional matrix can be improved, first of all by involving other qualities which, after the introduction of more "intelligent" systems, have proven essential in computer use and are more and more acknowledged in computer culture. Imagination, to give just one example, plays an important role in programming, as well as in running programs or adapting programs to new functions. The multidimensionality of such matrix representations corresponds to the user's intellectual and emotional qualities (see Figure 19).

An immediate practical application of this representation is in designing interfaces for the handicapped. The way people with impaired hearing and vision and those with physical handicaps interpret signs belonging to language and other systems of expression has to be considered when interfaces meant to support their access to computers are designed. Although some research has been done, more is necessary. We know, for instance, that the deaf have prob-

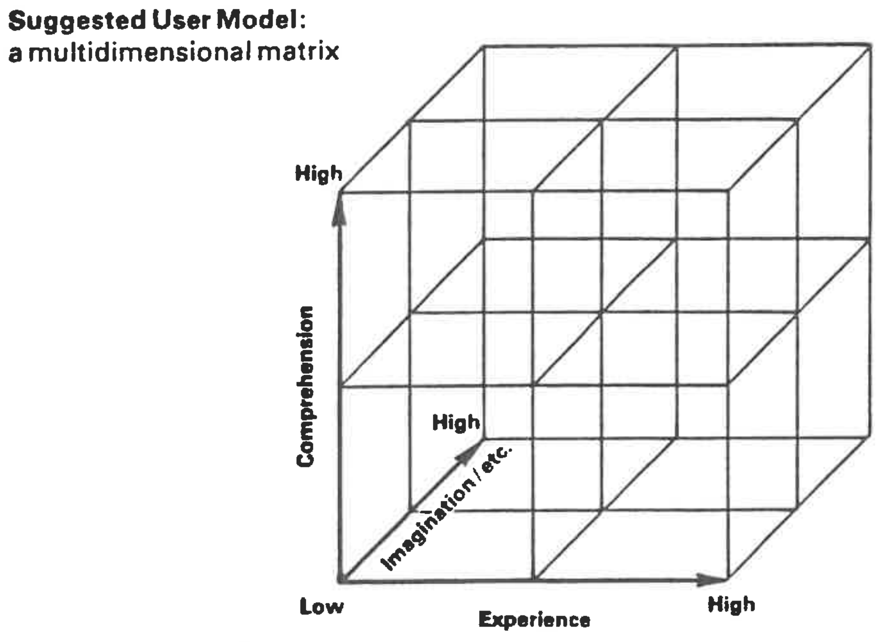


Figure 19. Suggested user model.

lems in dealing with time and time-related representations. Accordingly, interface has to be conceived as an expression of space relations. Audio components of user interfaces, on the other hand, help people with impaired vision. In this case, every quality of sound can be used, the pattern of sound sequences becoming part of the language. The semiotic concept of appropriateness applies in such cases.

COMPUTERS, USERS, AND COMMUNICATION

Signs are used predominantly for communication. They are means to represent our problems: mathematical signs for mathematical problems, 3-D representations for architectural problems, natural languages or formalisms (diagrams, vectors, matrices, etc.) for describing social problems. Computers are basically used for problem solving, a fact that should be carefully considered when communication issues are approached. As opposed to other tools, the computer is a "universal" problem solver once the problem is presented in a computable form (see Figure 20).

This model was devised after viewing computers from the semiotic perspective, i.e., after considering how people interact with each other through the intermediary of the messages they exchange. Bühler (1933), in his impressive semiotic study of social communication which, unfortunately, scientists outside his field know little of, and Jakobson (1960), in his research of linguistic processes, made

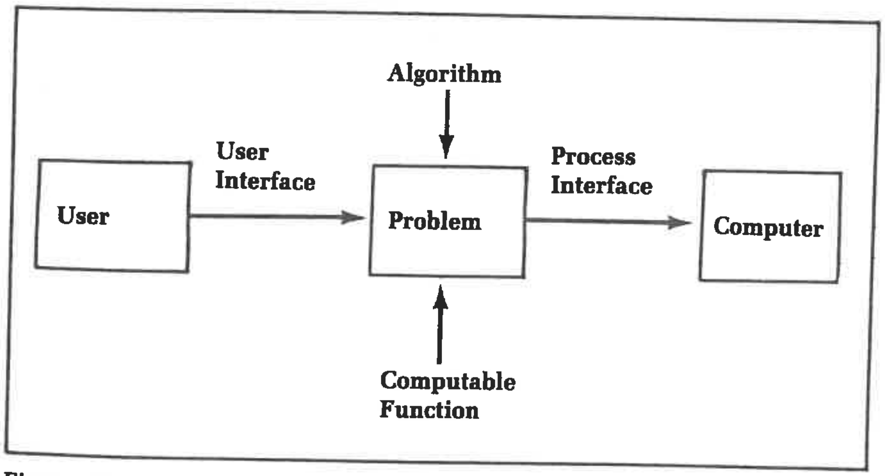


Figure 20. Semiotics of the user-computer relation.

contributions which help us better understand the nature of communication.

Examining this model (see Figure 21), (to which Calude and Marcus (1985) brought their contribution), we can identify several functions, to which I add their implications for the user–computer relation.

1. the function of communication—actually the function of maintaining communication, identified as the phatic function
2. the expressive function—relating addresser and the message
3. the metalinguistic function—dealing with the functioning of the code(s) used (expressing both the addresser–code and addressee–code relations)
4. the pragmatic function—dealing with the context and the way it influences communication (relation between addresser, addressee, and context)
5. the connative function—representing the attitude of the addressee towards the message (imperative messages are quite different from optional or query messages)
6. the design function—reflecting the way the addresser and addressee (in particular) relate to the medium
7. the referential (or cognitive) function—dealing with the meaning of the message
8. the formal (poetic) function—pertaining to the message's formal qualities (syntax error, for instance, is but an indication of this function).

Looking at a symmetrical communication structure, we have to

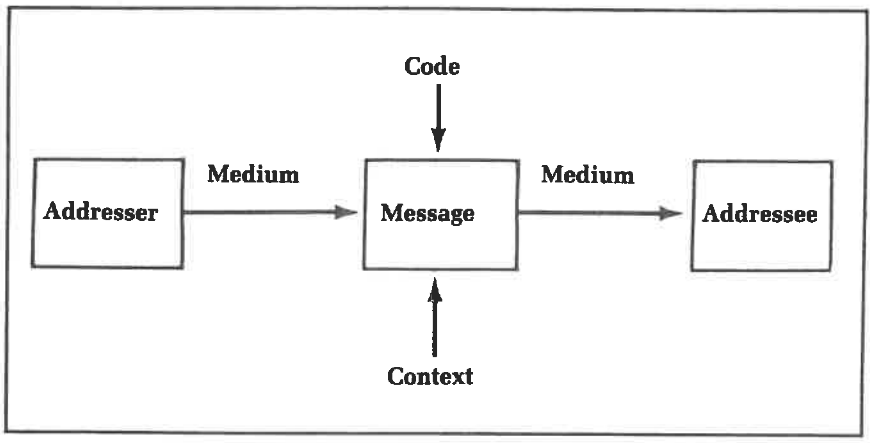


Figure 21. Communication in the semiotic concept (cf. Buhler-Jakobson).

consider specific situations related to the three basic segments identified in order to obtain optimal design: user–computer, computer–computer, computer–user. Examining Figures 20 and 21, the reader will notice the line of thought pursued: The message is the problem to be solved with the computer's help; the context and the code are represented by the computable function which describes the problem and by the program as based on some known or newly developed algorithm. A minimal requirement is that communication be maintained (the phatic function). This minimum proves quite complex (ever heard of crashing?) and involves the relations among user–computable function, computable function–program, program–computer, user–computer. Some computable functions describe a given problem better than others. But not all are equally computable; and if in principle they are computable, then some limitations in the hardware may affect the response time. And since each system comes with specifications impossible to avoid, whether the given system can accept the program becomes an issue. And if it can, how effectively? Finally, the messages meant for the user (and "issued" by the computer) should be concise, precise, and understandable—conditions easier to claim than to implement. I shall refer here only to the connative function, from among the others, mainly because interface issues are concerned with the type of problems a system is supposed to assist the user in solving. Two different forms of "intelligence" are evident: the "intelligence" built (wired) in the hardware, and the "intelligence" of the program. Several design decisions are expected in regard to error handling (interface and compiler, or interface of an environment like LISP or PROLOG), feedback to the user (how? what? why?), type of processing (effective, virtual), etc. The referential function is difficult to approach because, while the computer is a Boolean machine, the relation between the concrete problem and the computable function can be described in modal, not binary, logic. The expressive function, influenced by the same two-valued logic, reflects the state of the art in deterministic thinking, hopefully to be improved by the use of fuzzy logic or of the logic of vagueness. Zadeh (1984) is the best source for more information on this matter. Looking from the computer to the user, we see a slightly different picture in Figure 22. Obviously, the referential function of this segment is the same as the metalinguistic function of the user–computer segment. Two interesting aspects relate to the expressive function:

1. As a Turing machine, the computer can deal with the com-

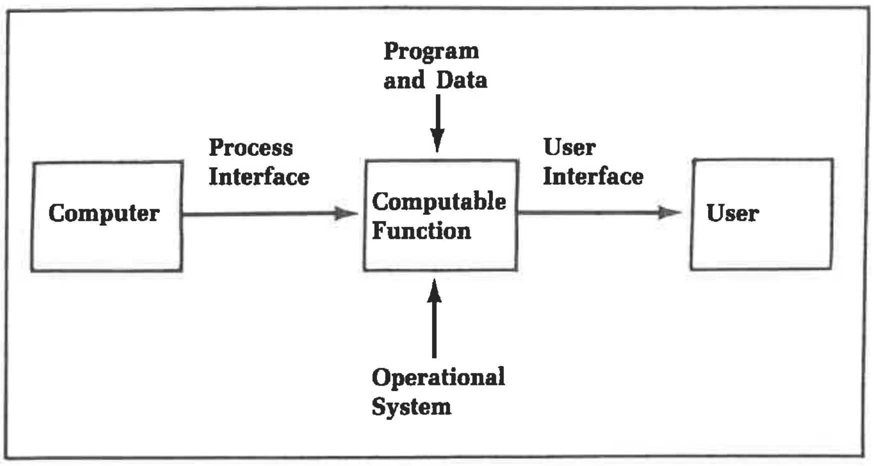


Figure 22. Computer-user relation.

putable function step by step (one thing at a time); i.e., no evaluation of the entire function is possible.

2. Moreover, the computer evaluates only a limited part of the generally infinite function, which brings into discussion the so-called approximation of the infinite by the finite (in computer terms, the evaluation of algorithms by machines).

Recently, artificial intelligence concepts (Reichman-Adar 1984, pp. 157–218) have suggested ways to improve this function. The problem to be approached in this respect is the presentation of a computable function in machine language. Operationally satisfactory definitions for computable functions are far from being a trivial issue. The designer of interface (process interface in this case) should be aware of the semiotic implications of this issue. We can refer to compiler-related aspects as a particular case pertaining to the same segment of communication (see Figure 23). Very relevant here is the metalinguistic function, since what actually goes on is "translation" (from programming language to machine language). We refer to three semiotic aspects of such translations: Is it faithful? How complex is it? How efficient is it? Although the user is not distinctly referred to in this segment, the formal (poetic) function is very important. The programming language influences the way the search for syntactic errors takes place and, in the case of more advanced systems, the so-called level of gravity (permissiveness of the system). Some languages better support this function by allowing a higher level of gravity (that is, although a program may have some errors, it is "accepted" in the processing phase). More recently developed programming languages provide an improved formal function.

The last segment to be considered concerns the computer user (see Figure 24). Basically, this segment deals with the way the results of computing (finite subset of the range of the computable function used) are made available/communicated to the user (assuming that the program was accepted and run and that the data was compatible with the software requirements). Semantic considerations are prevalent in this segment. The user ignores the metalinguistic function if the program performs well. In a debugging mode, this function becomes very important. As I have already shown, interface designers treat application environments and programming and debugging environments as though they were totally independent.

In short, the model presented here introduces normative concepts which interface designers should observe. It also suggests that evaluation should take place more systematically, submitting by explicit requirements a set of criteria derived from the semiotic perspective on which this model is built. Some requirements are already known (some even observed!); others have not been adopted in design interface. What is perhaps more important, the model explains the interrelation between the requirements, making clear that a system with high performance and low permissiveness—to mention one example—undermines the purpose of a user friendly interface. The vocabulary used in explaining the semiotic implications is of marginal importance. I do not suggest that designers learn this vocabulary, but that their understanding of what is explained by using

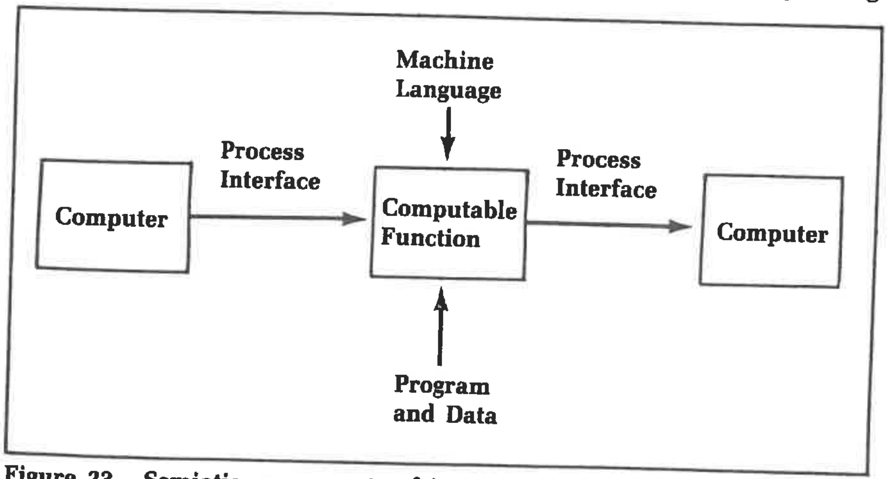


Figure 23. Semiotic components of inner computer processes.

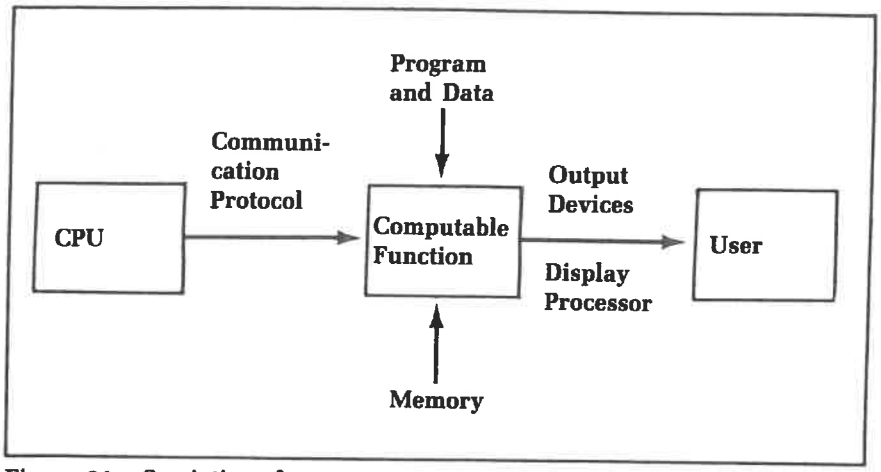


Figure 24. Semiotics of output.

these concepts can be improved. Entrenchment in jargon—and this holds true also for the computer jargon ("computerese")—annihilates the spirit of interdisciplinarity that we all agree upon.

ACCOUNTING FOR THE USER'S COGNITIVE CHARACTERISTICS

If we intend to deal with a user as concretely determined as possible, we have to incorporate results of cognitive studies of humans in the knowledge used for improved interface design. Some human characteristics are fuzzy: attention, response, adequacy of sign interpretation, to name a few. Others, while perhaps significant in work with tools different from computers, only marginally affect interaction between user and computer: endurance, sociability, exercise of physical strength, to name a few in this category. Attention should be given also to affordance parameters, reflecting environmental requirements and a better understanding of the relation human being–environment. Partially, ergonomic considerations cover such affordance parameters. In each of the above cases, semiotics is involved mainly in the way people externalize their opinions and beliefs and in how they express themselves and make their expressions understandable to others. In a matrix characterizing the future user, we should also deal with how human beings form their representations, beliefs, ideas, doubts, i.e., the internal semiotic characteristics of thought processes and emotions. Cognitive scientists agree with semiotic theories concerning the semiotic nature of thinking as well as the semiotic implications of feeling. One aspect of immediate practical relevance can be given as an example.

Whether the distinction left–right hemispheres of the brain can be sustained or not—an issue very much on the minds of psychologists and cognitive scientists—we cannot ignore the semiotic observation that signs can be structured in sequence (arrays of symbols) or configuration (e.g., visual constructions). The two modes in which we perceive and organize information are reflected in the characteristics of their interpretation. As far as we know, human beings process symbolic information mainly sequentially. Computers function the same way. Configurational systems of signs are processed by human beings in a parallel way. In the first case, a predominantly analytical dimension is apparent; in the second, a synthetic dimension. In sequential processing of signs, there is a dominant attempt to differentiate; in configurational, integration dominates. Time is related to sequence (our time representations are sequential), while space is related to configuration (see Figure 25). The two modes are interrelated, interfere with, or try to suppress each other; under certain circumstances, they enhance each other. To involve the user in a homogenous environment, i.e., to avoid abrupt switching from one mode to the other, is a minimal requirement almost consistently ignored by interface designers. Even when the designer provides a pointing device (e.g., a mouse), one type of user will rely on emulating keys in order to avoid swift changes that in several tests have proven exhausting (Patterson, 1983, pp. 75–82). A second requirement, reflecting the fact that users are so different, is to give the user a choice of dominant mode. Cooperation or interference between the basic cognitive modes takes place through both hardware and software (see Figure 26).

Physical properties (of the keyboard, display, printer, pointing device, etc.) are but an extension of the properties of the system in its entirety. While aesthetic and functional criteria are difficult to encode, they influence the design of the interface. No matter how attractively the mouse is designed, if the user is forced to switch often from pointing (configuration semiotics) to the keyboard (sequence semiotics), the design is inappropriate. To provide a really user-friendly interface means to make possible not everything, only what is acceptable. Aesthetic and functional acceptability, as well as cultural adequacy, are becoming ever more critical qualities once computer technology matures. Only a superficial designer, who targets the lower level of the market, thinks that cultural adequacy is reducible to emulation of characters used in foreign languages. Unfortunately, almost nothing is ever attempted beyond this. (IBM

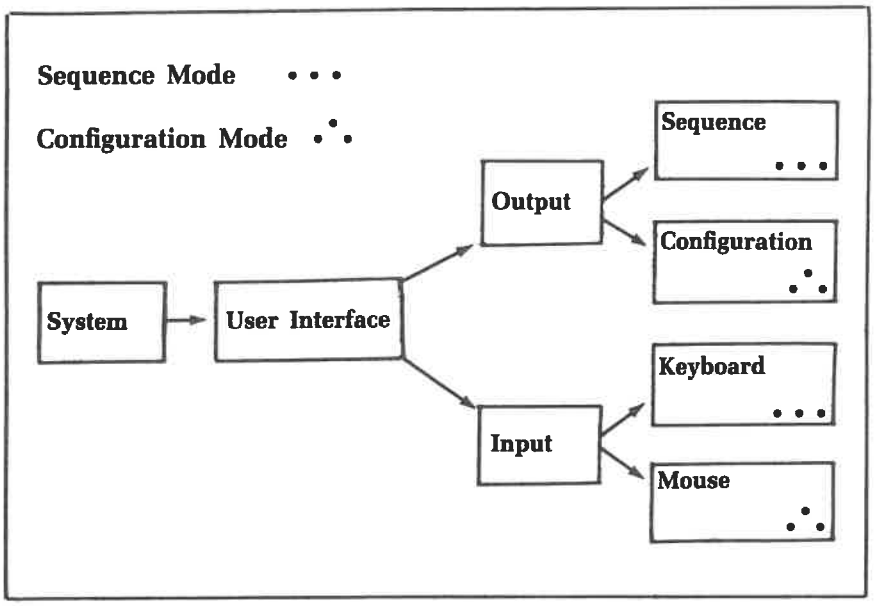


Figure 25. Sequence vs. configuration.

is a rare exception, AT&T a promising challenger, with impressive user interface accomplishments, especially on systems designed for internal use or switches). Typically, designers approaching interface issues, particularly communication aspects, are obsessed with quantitative aspects or make decisions based on intuition. Neither can be ignored, but to reduce interface issues to quantity is unacceptable.

An example of how this direction should be pursued is given by the Vivarium Project, a curriculum-driven project seeking to teach children about animals and initiated by Alan Kay. In her paper, Allison Druin (1986, p. 2), a former student of mine, currently a graduate student in the MIT Media Lab, asks: What will the next generation of computer learning tools be? From among many semiotic aspects of the Vivarium, Allison Druin (1986, p. 2) concentrates on one alternative (the furry computer called Noobie) to the traditional terminal: "a soft and inviting environment, which surrounds the child, and looks like a five-foot version of the squeezable input device . . . what emerged was the idea of a tactile computing environment in which the child would sit and create animals in the actual input device." We notice here several ideas relevant to the approach: the "universal" tool can be adapted to various tasks, not only through its application programs—which is the dominant strategy today—but also through an interface design adapted to the

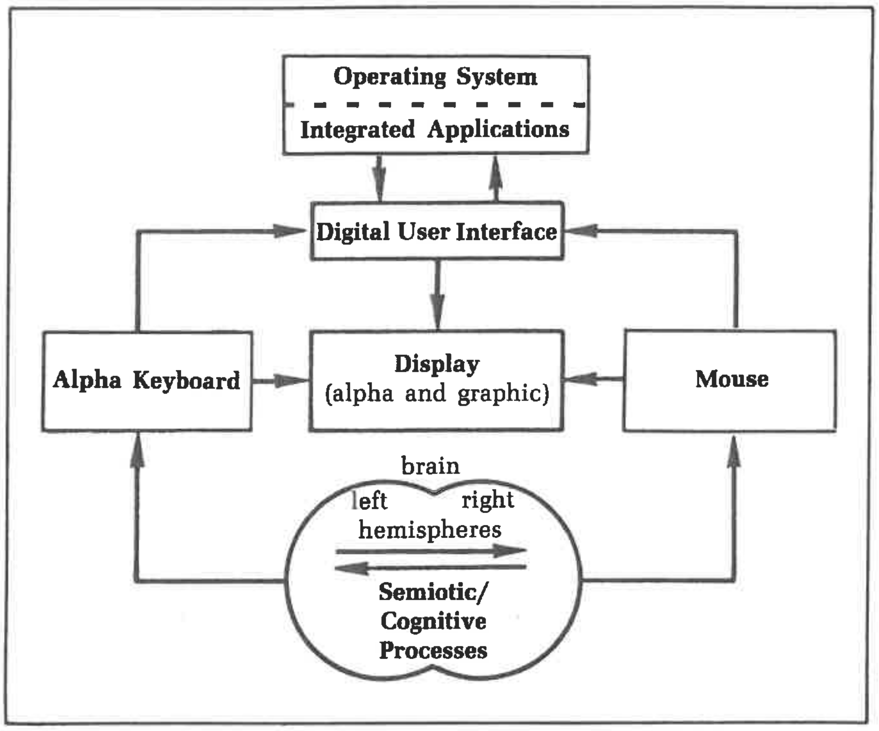


Figure 26. Integrating sequence and configuration.

cognitive characteristics of the activity that the computer will support. Adaptation of the tool to the problem will take place through the intermediary of different interfaces that are part of the system and that simultaneously connect the system with other environments: the user, other systems of education such as toys, stuffed animals, drawings, animation, entertainment. (It is no accident that the Henson Group is involved in this project.) Programs are abstract entities which obey formal rules. Applications, such as editing a document and retrieving from a database, cause an abstract entity (document, database) approachable through appropriate interface to have a concrete reality: Documents receive names; information can be ordered and formatted according to needs.

In short, using a computer means to make the abstract concrete. As opposed to typewriters, a word processing program is *all* the typewriters which can exist. A database program has the same characteristic. The designers of the Vivarium understood this semiotic peculiarity of computer tools, extended the limit to which the abstract program is made concrete, and integrated even the terminal and input devices. Indeed, the study of animals by children cannot be reduced to an exercise in typing on the typical keyboard, or to a display, no matter how sophisticated, on a CRT screen. There is more to the experience of such a class. There is the sense of touch, the sense of proportion, sound, etc. Correspondence to the real world will be ensured through an improved convention of likeness. This treatment of interface makes several channels of interaction necessary.

PRESENT AND FUTURE

The different ways users interact with interface devices is very important and should be accounted for in design specifications. But while we understand how the machines we build work, and even manage to find out why their functioning sometimes seems "irrational," we only partially understand processes in which our thinking and emotions are involved. Some progress has been made in understanding behavioral aspects; cognitive processes have been extensively and intensively researched, too. The results are frequently applied in the design of interfaces. The following aspects are routinely observed: message from user to computer, feedback, and computing and return of results. But, as was already mentioned, interfacing goes well beyond these and extends to everything a user will come in contact with when using a system and getting output from it (on CRT display, slide, film, hardcopy, etc.). Two attitudes regarding how interface should be approached can be identified:

1. Emulate the current human way of thinking and acting on the computer. ("It is important that the formal computer procedures do not prevent the user from changing his representation of the problem or task environment necessary to reach the best solution."*)
2. Challenge the user with a totally new language, thus with a totally new way of thinking and acting.

In both cases, a better understanding of what languages are, how they are used, and how they work is necessary. Our expectations are reliability, with tolerance towards the user if possible, self-sufficiency, ease of use, and adaptability. All relate to the semiotic qualities of interface language. We can distinguish between as many

* This is a quotation whose source I no longer know.

languages, as many senses, as we have; and this distinction encouraged the Vivarium concept. A "taste" statement can go as a mixture of, or succession from, sour, bitter, sweet. . . . hot, warm, cold . . . or can represent an example of a touch-interpreted statement. The same holds for smell. Programs to take advantage of this have been written or seriously considered. Van Dam (1984, p. 646) confirmed this when he stated, "The computer interface may eventually metamorphose into a total sensory environment." This being the case, we have to prepare ourselves for the challenge of the task by understanding the specific semiotics of each sensory environment through paying attention to general semiotic principles.

The main system of signs are the visual and the verbal (natural language). In reality, the distinction is less obvious, and influences between both are very important. Another distinction is between natural and formal languages. A suggestive representation can be given as a matrix (see Figure 27). Voice input devices, or other I/O features (heat sensitive, touch sensitive, for instance), will of course require a more sophisticated matrix which is not just multidimensional but also reflects influences between the different components. The distinction of visual–verbal first of all refers to possible forms of representation. The distinction of natural–formal refers to the logical structure and thus to the nature of language (cultural vs. artificial). The matrix takes combinations into consideration, too. Formal verbal languages may prove difficult for people to read or write in. Such languages require special training and a competence level demanded by specialized fields (mathematics, symbolic logic, language programming, dance, music, etc.). Reading text written in a programming language, or a musical score, or a choreographic laba-notation is difficult. Formal visual languages may prove difficult to "write" in but can be read more easily (not necessarily with precision). Research has recently approached such languages and the attempts to resuscitate visual modes based on pictographic representation or to improve such forms of visual representation (e.g., diagrams, charts, lists, etc.). Although controlled by grammar, natural languages are easier to write in because this grammar is not as rigidly specified as the grammar of formal languages. On the other hand, it is harder to be specific, precise, and to avoid ambiguity in natural language. In the cultural environment, this is an advantage evidenced by qualities which are usually not duplicated in formal languages. This is not to say that natural language is easier to use, as so many assume. Bar-Hillel (1970) maintained that natural languages are essentially pragmatic-free. Whether something fundamental has changed since 1970 in respect to our understanding of the pragmatics of formal languages, or to the way such languages are used, is a matter of controversy. Nevertheless, the pragmatics of natural languages is far more difficult than the pragmatics of any other language (formal included). What we address here are issues of language use in two different environments: one in which the user is comfortable, since it is the environment of his or her everyday life; and the second, in which the user faces something less familiar, in which interface should play a mediating role. "The ideal situation," as van Dam (1984, p. 646) described it, in tune with many computer scientists and/or science-fiction writers, "would be to interact with the computer as if it were a helpful human being, perhaps chatting in natural language."

Progress in better emulating natural languages (English basically) is to be expected, but the use of natural languages can become possible only on computers applying the logic of such languages. In reviewing this paper, Tom Carey tried to convince us that there is no reason why multi-valued logic cannot be simulated. Interface is a trade-off in which amount and type (of signs used) are the fundamental parameters. Norman (1983, p. 1), who introduced a remarkable quantitative method for trade-off analysis, makes the basic

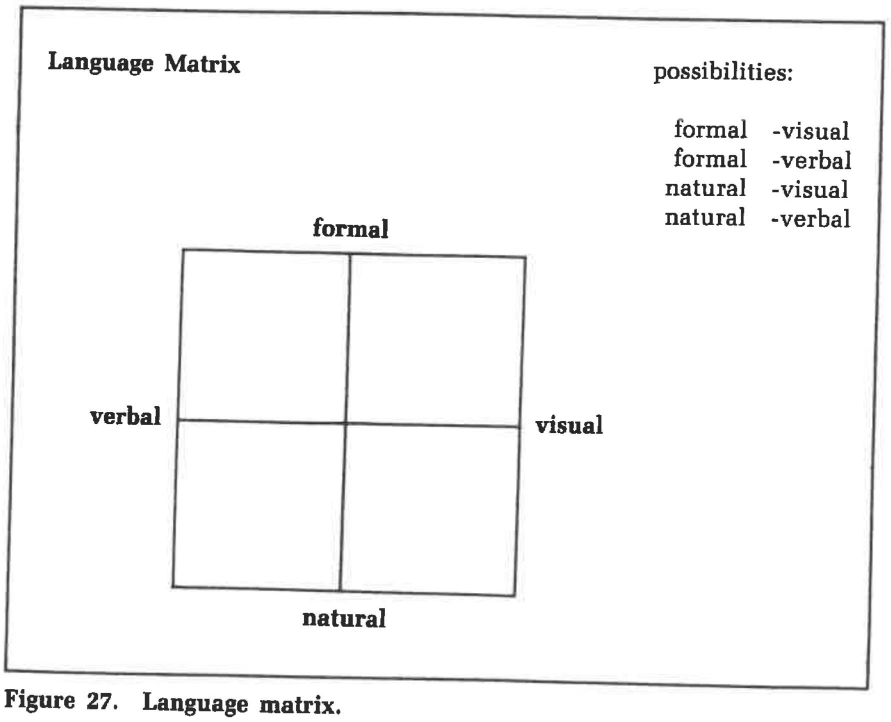


Figure 27. Language matrix.

statement: "Any single design technique is apt to have its virtues along one dimension compensated by deficiencies along another." Maybe Figure 28 will explain the kind of trade-off implicit in the semiotic decision involved in the design of interfaces:

In the Programmer's Hierarchical Interactive Graphics System (also known as PHIGS), goals specific to rendering visual presentation of information require a semiotic structure which supports interaction. As a toolset, the system permits model building and manipulation. Its transformation pipeline starts with visual primitives (modeling of coordinates) on which operations corresponding to viewing (from world coordinates to viewing coordinates, according to the photographic camera metaphor), clipping, and mapping are applied. The workstation independent elements are interfaced with workstation dependent elements. The organization of data is multilevel. Obviously we have here an implementation of a formal language in which the predefined grammar applies to images. As an interactive graphics interface, it supports a semantics specific to images and not to natural language statements. PHIGS is an integrated, interdisciplinary approach to interface which considers the contribution of each component. Semiotics coordinates the relation between everything that participates in interfacing. Product design, software engineering, hardware, ergonomics, etc.—highly specialized fields—should each in turn be evaluated and integrated in the comprehensive language of the product using PHIGS. Of course, semiotics has to provide the necessary means required. Recent

Natural Language	**Formal Language**
general	specific
unlimited vocabulary	limited vocabulary
indefinite grammar	predefined grammar
intuitive structure	logical structure
easily acquired competence	difficult to acquire competence
difficult to obtain performance	easy to attain high performance/ easy to learn

Communications Characteristics of Formal Languages

write	**select**	**read**
understand system	read system	model system
concrete knowledge	intuition	intuition
have purpose	have purpose	indefinite purpose
syntax error	semantics error	semantics error

Figure 28. Natural vs. formal languages.

contributions (cf. Winograd and Flores, 1986; Lakoff, 1987) concerning conception of the mind and understanding of computers suggest alternative strategies for designing computer systems. It seems that the maturing of technology and difficulties in accounting for failures affecting people's use of computer technology made the need for improved understanding of human–computer interaction more critical than at any previous time.

FINAL REMARK

In concluding one of the most passionate debates of the CHI '86 ACM-SIGCHI Conference on Human Factors in Computing Systems (Boston, April 1986)—the one between Stu Card and Don Norman—moderator Richard Pew noted that it explored whether "systematic effort applied to interface design is a worthwhile effort." This chapter contributes to an affirmation without ignoring that "design through natural evolution" (comparable to the natural evolution of our use of various signs) brings its own contribution. Method helps intuition if it is not transformed into dictatorship. Intuition augments method if it does not instill anarchy. In every moment of our semiotic existence, method and intuition complement each other.

ACKNOWLEDGEMENTS

The author is grateful to Leif Allmendinger, Thomas Ockerse, Richard Zakia, and Harvey Carapella for valuable comments and for providing some of the figures used here. In addition, the author thanks his students from the class, "Semiotics of Computer-Aided Human Activity", held at The Rochester Institute of Technology in Spring, 1984, during his tenure as William A. Kern Institute Professor in Communications, and "Computer Graphics-Graphics Design Issues," Autumn 1984, RISD. The reviewers, Dennis Wixon and Tom Carey, as well as the editors, Rex Hartson and Deborah Hix, provided criticism and questioned many of my views. If the chapter improved and is at the level of their expectations, I gladly share with them in the credit it deserves. Shortcomings are, of course, my responsibility.

REFERENCES

Bar-Hillel, Y. (1970). Communication and argumentation in pragmatic languages. In Y. Bar-Hillel (Ed.), *Pragmatics of natural languages.* New York: Humanities Press.

Bühler, K. (1933). Die Axiomatik der Sprachwissenschaft. *Kant Studien, 38,* 19–90.

Calude, C., & Marcus, S. (1985). Introduction to the Semiotics of Man-Computer Communication. In M. Nadin (Ed.), *New elements in the semiotics of communication.* Tübingen: Gunter Narr Verlag.

Druin, A. (1986). *A Vivarium project* (MIT Media Laboratory Working Papers). July. Cambridge, MA.

Goldberg, A., & Robson, D. (1979). A metaphor for user interface design. *Proceedings 12th Hawaii International Conference System Sciences, 6*(1), 148–157.

Haber, R.N., & Myers, B.L. (1982). Memory for pictograms, pictures, and words separately and all mixed up. *Perception, 11,* 57–67.

Jakobson, R. (1960). Linguistics and poetics. In T. Sebeok (Ed.), *Style and language.* Cambridge, MA: MIT Press.

Ketner, K.L. (with the assistance of A.F. Stewart) (1984). The early history of computer design. Charles Sanders Peirce and Marquand's logical machines. *The Princeton University Library Chronicle, XLV* (3), 187–225.

Lakoff, G. (1987). *Women, fire, and dangerous things. What categories reveal about the mind.* Chicago/London: The University of Chicago Press.

Marcus, A. (1983, July). Graphic design for computer graphics. *IEEE, 3*(4) 63–69. Los Alamitos, CA: True Seaborn.

Marcus, A. (1983). Designing iconic interfaces. *Nicograph 83,* pp. 103–122. Tokyo: Seminar Notes.

Meyrowitz, N., & van Dam, A. (1982). Interactive editing systems. *Computing Surveys, 14*(3), 323.

Nadin, M. (1981). *Zeichen und Wert* [Sign and value]. Tübingen: Narr Verlag.

Nadin, M. (1986). Design machine. Columbus: Art and design technology research notes.

Norman, D.A. (1983). Design principles for human-computer interfaces. Human Factors in Computing Systems CHI '83. *Proceedings.* New York: ACM.

Patterson, M.L. (1983, November). Graphical interface design considerations. *Computer Graphics World, 6*(11), 75–82. Tulsa, OK: Pennwell Publications.

Peirce, C.S. (1932). *The collected papers of Charles Sanders Peirce* (C. Hartshorne & P. Weiss, Eds., Vol. 2). Cambridge, MA: Harvard University Press.

Reichman-Adar, R. (1984). Extended person-machine interface. *Artificial Intelligence, 22,* 157–218.

Schneider, M.L., & Thomas, J.C. (Eds.). (1983). Introduction: the humanization of computer interfaces. *ACM Communications 26*(4), 252–253.

Shannon, C., & Weaver, W. (1947). *The mathematical theory of communication.* Urbana, IL: University of Illinois Press.
Simon, H. (1982). *The sciences of the artificial.* Cambridge, MA: MIT Press.
Tesler, L. (1981). The Smalltalk environment. *Byte,* August, p. 90.
Thacker, C.P., McCraight, E.M., Lampson, B.W., Spronll, R.F., & Boggs, D.R. (1979). *Alto: A personal computer* (Report CSL–79–11). Palo Alto, CA: Xerox Palo Alto Research Center.
van Dam, A. (1984). Computer graphics comes of age [interview]. *ACM Communications, 27*(7), 646.
Winograd, T., and Flores, F. (1986). *Understanding computers and cognition. A new foundation for design.* Norwood, NJ: Ablex Publishing Corp.
Zadeh, L. (1984). Coping with the impression of the real world [interview]. *ACM Communications, 27*(4), 304–311.