Interface design: A semiotic paradigm*

MIHAI NADIN

Design principles are semiotic by nature. To design means to 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, etcetera. Design comes about in an environment traditionally called *culture*, currently identified as *artificial* (through a rather romantic distinction between natural and artificial), and acts as a bridge between scientific and humanistic praxes. Along this line of thinking, Simon (1982) stated, 'Engineering, medicine, business, architecture, and painting are concerned not with the necessary but with the contingent — not with how things are but how things might be — in short, with design'. The object of semiotics is sign systems and their functioning within culture. For a long time (and for reasons whose presentation is beyond the scope of this article), one type of sign — the symbol — has been considered representative of *all* signs in human culture: 'for most of us ... the significant part of the environment consists mostly of strings of artifacts called "symbols" that we receive through eyes and ears in the form of written and spoken language and that we pour out into the environment — as I am now doing — by mouth or hand' (Simon 1982). Actually, we perceive signs through all our senses, and we generate signs that address the same. The fact that some of these signs (visual, auditory) are more important should not prevent us from considering any other sign that can be used for representation, communication, and communication functions. But before dealing with these basic functions, we have to settle upon one of the many definitions of sign that have been advanced in the field of semiotics, and then apply it as consistently as possible. The definitions fall into two basic categories:

1. Adoption of one kind of sign — usually pertaining to verbal language — as a paradigm, with the understanding that every other sign is structurally equivalent. Artificial intelligence researchers are quite comfortable with this model. The Swiss linguist Ferdinand de Saussure advanced

Semiotica 69–3/4 (1988), 269–302. 0037–1998/88/0069–0269 $2.00

the definition of signs as the unity between a *signifier* (the actual sign embodied in some material form such as words, shapes) and the *signified* (what the sign is supposed to mean).

2. Adoption of a logical structure, with the understanding that each type of sign and each sign operation can be described within a pan-logical system. The American scientist and logician Charles S. Peirce (1839–1914) advanced the definition of sign as 'something that stands to someone for something in some respect or capacity' (1931–1966). No matter which definition is adopted, the question of semiotic laws governing sign processes is necessarily raised. Remaining within the realm of sign as symbol, Simon felt entitled to state, 'The laws that govern these strings of symbols, the laws that govern the occasions on which we emit and receive them, the determinants of their content are all consequences of our collective artifice' (1982). Both Saussure and Peirce described the same through the role of the social, a semantic equivalent of 'collective artifice'. Although Simon is mistaken in limiting the sign to the artifact — we can and do interpret semiotically (that is, as a sign) natural occurrences, too — he is correct in considering signs as having an air of contingency, as natural phenomena having an air of necessity, in his opinion. For several reasons, the pan-logical definition of the sign is more appropriate to the subject approached here, not only because the underlying principles of computers are themselves logical, but also because design activities are not reducible to the model of verbal language (or of any other sign system). On the basis of Peirce's above-mentioned definition, this visual representation (not the only one possible) can serve as an operational model (Figure 1). Figure 1 should be read as saying that only the *unity* between the three components represents a sign — that is, that signs are

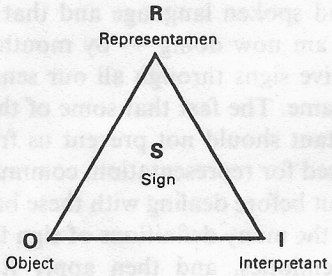

Figure 1. *Sign.* S = S(O,R,I); Representation = that which represents; Object = that which is represented; Interpretant = the process of interpretation

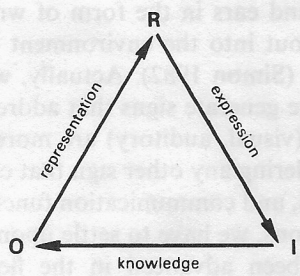

Figure 2. *Semiotics.* Semiotics as science of representation; semiotics as science of expression; and semiotics as science of knowledge

identified as such only through their representation, and that as soon as we interpret a sign, we become part of it for the time of that interpretation. The functions of a sign are also evident in Figure 2.

Semiotic levels at which sign processes (semioses) take place, levels that are undoubtedly familiar and important in computer science, can also be depicted (see Figure 3).

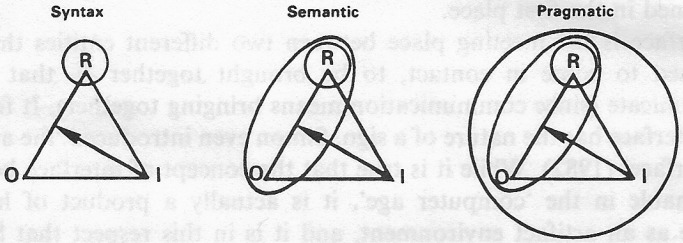

Figure 3. *Semiotic levels of semioses.* Syntax = the relation between signs, how signs are constituted; Semantic = the relation between sign and object, what the signs are conveying; and Pragmatic = the relation between signs and the user, what signs are used for (cf. Morris, 1938)

There is little trouble in understanding from this that no sign can be considered independent of its relation(s) to other signs, whether similar (such as words in a given language) or different (words, images, sensory perceptions, etcetera). The interdisciplinarity of semiotics is a consequence of the fact that sign processes are heterogeneous by their condition, and that in order to understand how different kinds of signs constitute interpretable strings or configurations, we have to become acquainted with each different kind, as well as with the principles governing human or machine interpretation of such strings or configurations. Representation of an object, and the consequent interpretation of such a representation, can take three different forms (Figure 4).

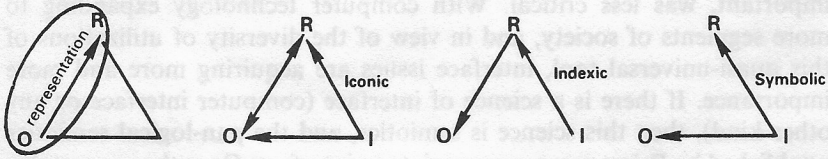

Figure 4. *Representation.* An object can be represented Iconically = representation based on resemblance, likeness; Indexically = representation causally influenced by the object, mark of the object; Symbolically = representation based on convention

It should by now be clear why Simon's concern with symbols alone (which is also the concern of the field known as symbolic anthropology) proves a serious limitation of his explanatory model. However, since symbols are the dominant sign representation in human culture, and since each symbol contains iconic or indexical elements, it is easy to reformulate some of Simon's ideas in order more adequately to make use of the semiotic principles governing those cognitive phenomena with which he is concerned in the first place.

Interface is the meeting place between two different entities that are supposed to come in contact, to be brought together — that is, to communicate (since communication means bringing together). It follows that interface has the nature of a sign. Simon even introduced 'the artifact as interface' (1982). While it is true that the concept of interface became fashionable in the 'computer age', it is actually a product of human culture as an artifact environment, and it is in this respect that Simon regarded 'The artifact as interface' and 'The environment as mold' (1982). Interface is also a problem of human-to-human relations, especially in the context in which human contact and interinfluence become more and more *mediated.* In defining the sign as a mediating entity and semiotics as the theory and practice of mediation, I suggest that despite the diversity of signs and sign processes, these all fulfill the basic function of *intermediary*, *go-between*, *medium* between two or several distinct entities brought together through human activity. The contingency of each mediation — its *likelihood*, *relative unpredictability*, its *dependency* on and *conditioning* by other factors (that is, the contingent nature of each interface) — is a reflex of design's double nature as science (in respect to the scientific principles of design) and art (in respect to a particular, original way of designing). Schneider and Thomas (1983) pose two questions that express the feelings of the computer community: (1) Why isn't the design of computer interfaces more like science? and (2) Why can't the people who design interfaces be more like engineers? These concerns stem from the recognition (late, but nevertheless a recognition) of the role interface plays in the human use of computers. In the past, interfacing, although important, was less critical. With computer technology expanding to more segments of society, and in view of the diversity of utilizations of this quasi-universal tool, interface issues are acquiring more and more importance. If there is a science of interface (computer interface or any other kind), then this science is semiotics, and the pan-logical semiotics established by Peirce seems appropriate to interface. Once they accept this affirmation, computer scientists and engineers should have no problem in understanding the multifaceted nature of semiotics as *science and art*, *heuristics and hermeneutics*, etcetera. Programming, while a very rigorous

activity, allows for creative algorithms and creative interpretations of algorithms. Two programs for the same activity can be as original and innovative as their authors. The scientific nature of logic, reflected in the scientific nature of the computer, implies the *art of reasoning* and allows for an *art of computing* expressed in elegant, balanced, optimized codes. Since the recognition of the fact that computers are basically sign processing devices, not merely number crunchers, progress has been made in freeing ourselves from the naive assumption that all we have to do in order to achieve intelligent actions through devices is to duplicate the structures of the human brain (the McCulloch–Pitts line of thought). The functionalist (cognitive) paradigm states that essentially, software is to hardware what mind is to brain. This implies that thinking processes are sign processes. (A discussion of this presemiotic paradigm is beyond the scope of this article.) All that we know, we know through the intermediary of signs and *in* signs; and all that we apply from our knowledge is semiotic in nature.

Based on these elements, I shall introduce a generalized concept of interface and then apply it to actual computer systems. First, I should point out that interface, no matter what kind, specifies the optimal set of signs for the interaction between two entities, whether 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 upon which that particular system is based (see Figure 5).

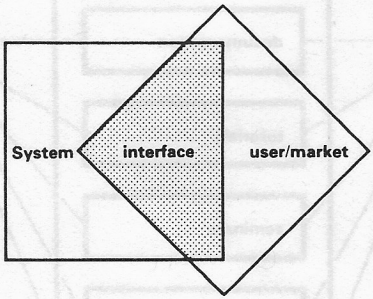

Figure 5. *Interface*

Cars, radios, dishwashers, vending machines, etcetera all require interfacing in order to be optimally used. Each requires a certain sequence of actions that allows for the pragmatics of using it. In the case of computers, Meyrowitz and Van Dam (1982) ascertained that user interface, together with the conceptual model, constitute the interactive editor.

According to this conception, user interface comprises 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, etcetera). By extension, a manufacturer of computers interacts with the market through numerous constitutive interfaces or, as I shall argue, through the *language* of the product and of everything participating in its marketing (see Figure 6).

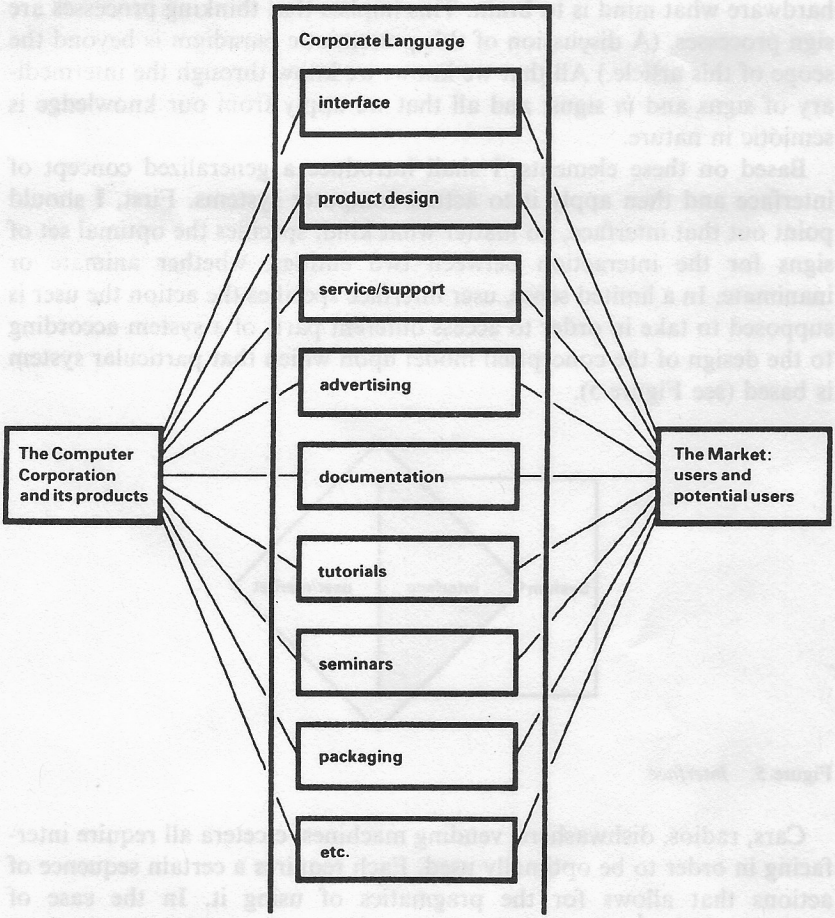

Figure 6. *Generalized interface model*

What makes things a bit more complicated in comparison to the most common social forms of interfacing through the intermediary of natural language (the most complicated semiotic system that we are aware of) is the fact 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 within the sphere of user interface activity. Accordingly, user interface should allow for the personal user model to be formed while the user learns the language of the interface. In the spirit of Chomsky's generative grammars, we understand language as a generative mechanism given in the form of a grammar to be applied to a vocabulary. (A strict mathematical definition is not necessary in this context.) Quite often, the personal user model differs from the conceptual model upon which user interface is developed. Although operationally different, the user model and the conceptual model should remain logically consistent. This is accomplished — when it is accomplished — by the semiotic system of user interface.

In order to give an idea of how semiotic methodology can be applied, I shall concentrate on a rather common example: the so-called office system computer. First of all, in order to proceed with the design, we have to identify the sign that constitutes the interface (or the language — see Figure 7).

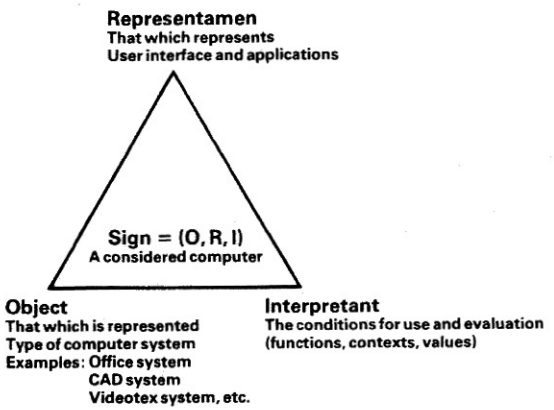

Figure 7.

The premise for considering a computer's interface from a semiotic viewpoint is that it represents a *complex sign system*, a *language*. More precisely, it represents a system we interpret as an emulation of *the office*.

The *pragmatics* results from the functions made available (word processing, ledger, listing, etcetera). 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. The final result that the designer and user look for, once the product becomes available, is not the value of true or false, as in formal logic, but *meaning*. This brings one more definition of semiotics to expression: semiotics is the logic of meaning. As such, it approaches the laws of sign processes meant to convey a certain meaning to an intended interpretant (that is, 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. An office is the unity of the *environment*, *tools*, *supplies*, and *activities* which make possible the execution of 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 (IBMs, DECs, Wangs), 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 (Table 1). Once a decision for specialization (insurance, financial planning, law practice, etcetera) is made, under the assumption that the production of a specialized computer is justified, the description becomes more specific. In other contexts (the European market, Far Eastern office activity, etcetera), the low-level protocol will look slightly different.

Sometimes, the way to proceed, especially when the visual component is desired for the intended interface, is to consider a visual representation of the office (see Figure 8). But an office is not a collection of files, typewriters, calculators, and so forth — it is not a collection of hardware, but essentially an *environment* where communication (exchange of documents, storage and retrieval of data, planning, etcetera) is possible and necessary. This is very important and explains why office systems successful in supporting individual office work but not communication have never really made it in the market. Lisa®[1] is an example that comes immediately to mind.

After defining the interface sign and specifying its elements with the aid of low-level protocol description, the next step is to define the type of representation (iconic, indexical, symbolic) and the type of command (prefix, postfix, infix). The two aspects are interrelated. Until the Palo Alto Research Center unveiled its then original iconic interface (the Alto®[2] station), the main type of command was the prefix. Basically, a prefix command specifies first the verb (operation) and then the object of the operation. It requires a predicative language as its interface language.

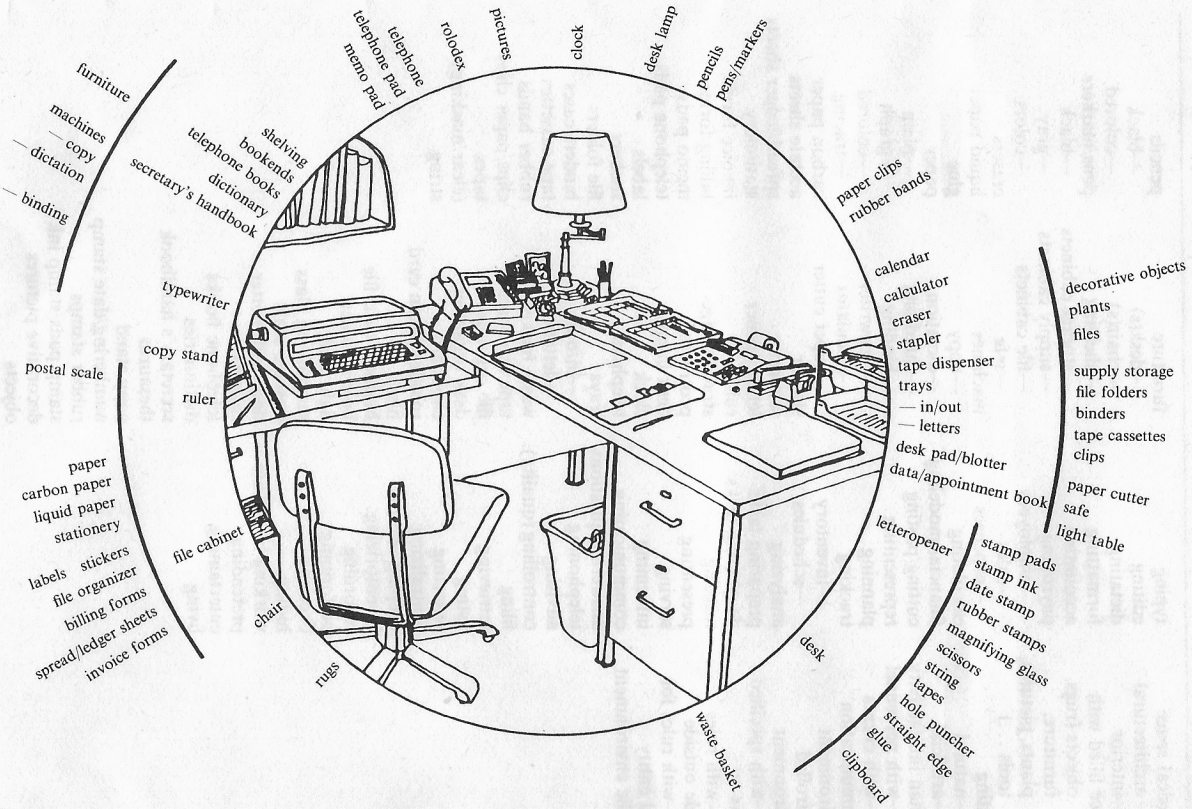

Figure 8. *Visual representation of an office*

Table 1. *Low-level protocol description of an office*

Environment	Activities	Tools	Supplies
physical space	typing	furniture	pencils
—architectural	editing	—desk(s)	—black
—interior	dictating	—chair(s)	—colored
space filled with	formatting	—shelving	pens/markers
objects (rugs,	accounting	—storage cabinets	—black
furniture,	payrolling	—supply cabinets	—gray
plants, pictures,	—employees	—file cabinets	—colors
tools ...)	—sales	—safe	erasers
lighting	—expenses	machines	liquid paper
—natural	calculating	—copy	glue
—artificial	financial modeling	—dictation	paper
human interactivity	cutting/pasting	—shredding	—plain
—with personnel	representing	—binding	—graph
—with clients	planning	—typewriter	—colored
communication	tracking	—calculator	—tracing
environment	—inventory	—paper cutter	carbon paper
controlled	—schedules	stapler	acetate sheets
environment	analyzing	scissors	spread/ledger sheets
—with specified	preparing tasks	letter opener	stationery
areas	developing tasks	rulers	invoice forms
—with rules	meeting	straight edge	billing forms
inside outside	presenting	protractor	memo pads
—with rules for	serving	desk lamps	telephone pads
legal entity	informing	clock	labels
public environment	communicating	telephone	stickers
	answering questions	trays	file folders
	telephoning	—in/out	binders/cases
	advising	—letters	tape cassettes
	controlling (quality)	waste basket	rubber bands
	filing	tape dispenser	clips/paper clips
	retrieving	file organizer	tapes
	listing	desk organizer	(clear/masking)
	reporting	clipboard	string
	centralizing	rolodex (open card	
	keeping records	files)	
	inventorizing	business card file	
	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	

The postfix command does just the opposite, allowing first for the selection of the object and then for the desired operation. It requires a subject-oriented interface language; that is, it makes possible a visual language. The infix command implies the existence of several operands, each action being virtually connected to such operands. Since the sequentiality of computer string processing is a structural given — and natural language is sequential — the first attempts to design interfaces accepted the sequential paradigm. The relation between the user and the computer typically followed the pattern of typing in text strings (as close to natural language as possible) for names (commands) and for operands. To ensure a certain feedback, the strings were (and still are) echoed on the output device (printer, screen) after being processed by the editor. Enough has been said about the limitations of this interface methodology and enough research has gone into improving it (function key, better emulation of natural language, 'intelligent' editors, etcetera) that it is not necessary to reopen the discussion here. The postfix command opened the way for menu-oriented interfacing — that is, for multiple choice from a number of strings and/or visual images (mainly iconic and representing objects or actions). Less has been said and less is known in respect to the limitations of such interfaces. For instance, in many actions and/or objects required to perform an intended computation, the distinction between object and action is not always clear-cut, response times sometimes increasing beyond the time of the operation itself. (A separate study is necessary to deal with such limitations.) Semiotics applied to the example of an office system allows for designing the set of primitives (visual, verbal) that will constitute the interactive editor language. Figure 10 displays a possible design strategy. What we see in Figure 9 is

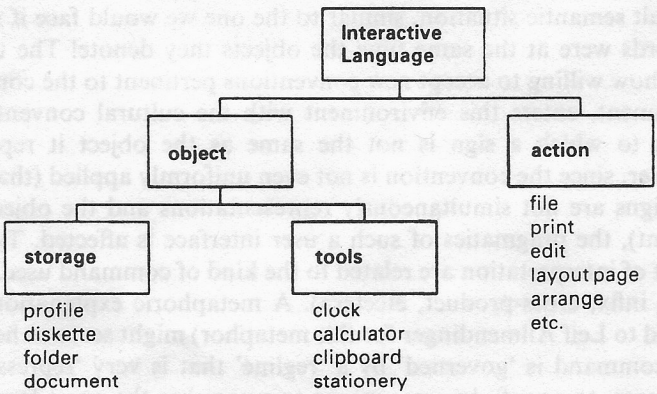

Figure 9. *Model of interface language*

that the machine's available computing capacity is supposed to support the generation, modification (addition, deletion, editing, etcetera), storage, retrieval, comparison, and so forth of texts, tables, diagrams, line art, drawings, or other computer programs. Actually, the three basic sign operations of *substitution*, *insertion*, and *omission*©[3] cover the entire gamut of operations the computer performs. (Simon gives a list: recoding, storing, copying, moving, erasing, comparing symbols within these one-thing-at-a-time systems.) What should be pointed out here is that the design of interface is a matter of *semiotic consistency* — that is, of uniformly using whatever means of representation are considered adequate. For instance, choosing visual representations of objects (as icons) and representations of actions through words (in pop-up menus) is a decision that makes sense only if implemented *consistently*. Some of the interfaces currently available lack this characteristic, exposing the user to awkward and confusing conventions. Interpretation is the issue here. Among the factors involved in the semiotic process of interpretation, the *amount* and *type* of signs interpreted play an important role. Basically, *amount* influences the *time* required to process (thought processes), while type affects the *kind* of processes. Several visual interfaces depict the calculator, for example (a simplistic option, but nevertheless a good example). The representation in Figure 10 visualizes the idea presented above.

Recognizing that the squares on an icon called 'calculator' are 'real' buttons involves a thinking process far from the common thinking adapted to the conventions accepted and culturally acknowledged in using a computer. 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 would face if some of our words were at the same time the objects they denote! The user, no matter how willing to accept new conventions pertinent to the computing environment, enters this environment with the cultural convention according to which a sign is not the same as the object it represents. Moreover, since the convention is not even uniformly applied (that is, the other signs are not simultaneously representations and the objects they represent), the pragmatics of such a user interface is affected. Type and amount of interpretation are related to the kind of command used (prefix, postfix, infix, cross-product, etcetera). A metaphoric explanation (I am indebted to Leif Allmendinger for this metaphor) might serve us here. The prefix command is 'governed' by a 'regime' that is very 'repressive'. In such a case, to specify lexemes means to remember the exact form of an ever increasing (and sometimes changing) set of commands, or at least

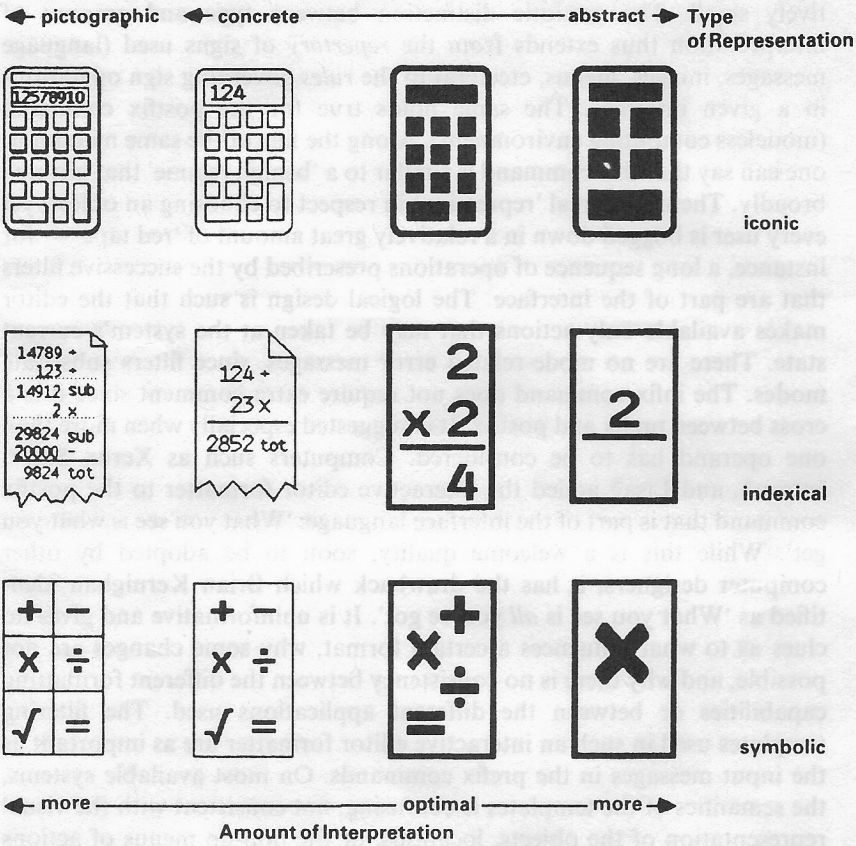

Figure 10.

their abbreviations (so different from one system to another). Even if prompts are provided when equivocal command abbreviations are used, the built-in restrictions cannot be overcome. 'Dissident' use is prohibited. Information or strings of signs typed in for processing, but 'prefixed' for nonsupported action, disappear without trace (since saving is usually possible after processing and before changing from one mode to another). IBM's CMS editor and DEC's SOS editor are offhand examples. Using function keys or special keys in 'shift' saves typing, but the system's permissiveness is not increased. The only friendliness comes through the error messages, sometimes so cute that the user becomes even more frustrated. Still, most use of the system according to the rules of the mode selected causes no interference in the user's routine. This command 'regime' governs 'brutally', although the amount of 'governing' is rela-

tively small. The semiotic distinction between type and amount of interpretation thus extends from the *repertory* of signs used (language messages, images, menus, etcetera) to the *rules* governing sign operations in a given interface. The same holds true for the postfix command (modeless computing environment). Along the line of the same metaphor, one can say that this command is similar to a 'benign regime' that governs broadly. There is no real 'repression' in respect to choosing an object, yet every user is bogged down in a relatively great amount of 'red tape' — for instance, a long sequence of operations prescribed by the successive filters that are part of the interface. The logical design is such that the editor makes available only actions that may be taken at the system's current state. There are no mode-related error messages, since filters substitute modes. The infix command does not require extra comment since it is a cross between prefix and postfix. It is suggested especially when more than one operand has to be considered. Computers such as Xerox Star®, Intran®, and Lisa® 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'. It is uninformative and gives no clues as to what influences a certain format, why some changes are not possible, and why there is no consistency between the different 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, not consistent with the visual representation of the objects, locations, or the pop-up menus of actions (based on an interface 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 conditon of raster graphics. It can be compensated either by increasing the density of pixels (which results in higher costs in computer memory) 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.

The field of human factors in computer systems, 'an unruly mixture of theoretical issues and practical problems' (Schneider and Thomas 1983), 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 not considered was the fact that signs and sign processing represent the common underlying principle of both human interaction with computers and of the computer, 'a member of an important family of artifacts called symbol systems' (Simon 1982). Since the technology upon and for which we build interface changes very rapidly, pan-logical semiotic principles, in their breadth and depth, provide a foundation for improved interface design (user and process interfaces), for instance in the design of software and hardware.

Since hardware issues are too often approached independently of future software applications, and since the integration of software is not possible without better adapted hardware design, I would like to deal with three aspects pertaining to this: principles to be observed in selecting components of interface language; aspects of man–machine communication (the semiotic paradigm versus the information paradigm of communication); and the 'language' of a computer and the various interfaces involved in the pragmatic user–computer relation. One thing should be made clear: 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). 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, and what weaves them is our use of signs, our *de-sign* (a spelling that emphasizes the intensive use of signs in design activities). 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; and (2) recognizing the whole and inferring from the whole to the parts. 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

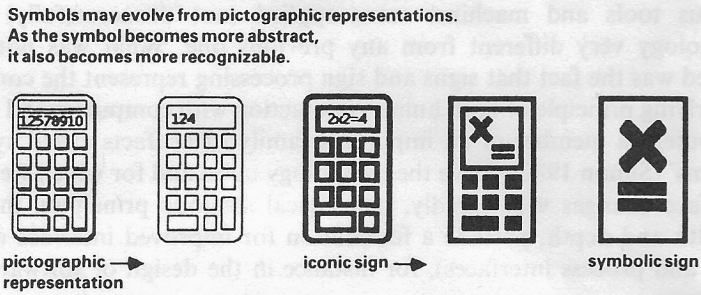

Figure 11. *Evolution of a sign representation*

(for instance, one of a solar battery calculator), he will have difficulties in 'using' the sign. The semiotic level is reached when the conventionality of the sign becomes evident. (Convention means here *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. 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. Once the user accepts a language, he will apply it according to the rules the designer embedded in the interface, and *their* communication, mediated by a certain machine, will take place. Obviously, understanding what an icon represents, as opposed to what it pictures, is essential for designing user interface language. (The diagrams in Figure 4 can be helpful in emphasizing what I have already explained about different ways of representing an object with a sign.)

Once this is understood and consistently applied, we can decide on one type of interface or another, or on 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. The example of the calculator (Figure 10) can again be helpful. Words can be used as well; and at the extreme of the symbolic representation, one can add the word *calculator* or some abbreviation.

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, omis-

sion) 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 a delayed addition to it. Despite its qualities, the Apple IIC (one among several possible examples) shows what happens when an interface concept is adopted primarily for marketing purposes. 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 (such as 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 *language* system. I would insist that designed interface language be, in principle, formal. This means that the language should function according to a logical structure which the user can 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, and user performance improves accordingly. 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 time-sharing. 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 pressures. 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, and to the languages of gestures, trade-

marks, etcetera. In extensions of user interface (documentation, tutorials, seminars, support, and so on), 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 actually represented by a divided cognitive structure, in which sequence and configuration (that is, time-related and space-related perceptions and activities) are not homogeneously supported by the brain, we should be able to design interface in such a way as not to affect the balance of these two basic cognitive modes.

Research in semiotics, within the general framework of the theory of learning, has made quite clear that comprehension of a specific system of signs means identification of the structure of that system. The 'transparency' of interface is not only a cognitive quality, but also an emotional quality, a fact impossible to ignore as long as we have human beings in mind when we talk about the user. In order to be made more apparent to the user, interface language should use (1) *concrete* representations of objects and storage, and (2) operation representations that relate *directly* to actions. Concreteness and directness must be expressed as clearly as possible. Sometimes the vehicle of language is better adapted to this exigency; other times images or sounds are more effective. Of course, the *proof of adequacy* is in the use as such. So-called *integrated* software packages attempt to support various activities. More often than not, such packages lack the ability to relate adequately programs which are good when taken independently but fail when combined because they were not conceived from an integrated semiotic perspective. Pseudo-integration becomes obvious in different instances of running such programs. Error and warning messages are examples of this unfortunate characteristic. Usually, an *alert file* concentrates these messages, but it is the routine of the alert manager designed to display them. Typically, an hour glass suggests waiting, despite the fact that in our culture the hourglass is a symbol for time, not for waiting. Error messages, usually anthropomorphized, reveal that each program was conceived independently, without any integration goals in the designer's mind. Confronted with a user who naturally integrates all his activities, manufacturers discovered that it is easier to 'unbundle' the 'integrated' package, then to 'improve' it (so they promise), than it is to 'redesign' it (from scratch), as would be necessary.

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, etcetera). Obviously, the design of such components (keyboards, tablets, light pens, painting devices, printers, and so forth) integrates product design considerations, ergonomy, psychology, marketing, etcet-

era. Once again, the unity of hardware–software and their reciprocal influence become critical. Needless to say, no system available today, as far as we know, was designed to integrate such diverse components. Dealing with a pointing device — like the 'mouse' — as an independent component (there are manufacturers that specialize in 'mice') means to contradict the basic requirements that ensure the consistency and adequacy of interface. There is nothing wrong with specialized manufacture — of 'mice' (mouses?), hard disk units, or other interface components — if a unifying and integrating design serves as premise. But this is rarely the case. What usually distinguishes such components is merely the trademark or, in better cases, the software used to interface them with the system. User interface language, together with all other components, is supposed to help in (1) pattern recognition (of one sign or a combination of signs); (2) associating signs during use of the system according to the designed syntax defined through the conceptual model; (3) expansion to other applications provided by the computer manufacturer (IBM, Digital, Data General, etcetera) or by independent software developers (which frequently ride 'piggyback' on a successful system).

A method developed specifically for uniformly dealing with all components and their interrelation is the *semiotic matrix*®[4] In this matrix, the basic user–system relation can be represented in an interdisciplinary, integrated way. The concept of interface is expanded to all those instances of the mediating effect of using sign systems/language for problem solving and/or communication. This expanded concept is not the mechanical result of examining the matrix of components, but a design that accounts for such components. A product's look and functionality are a continuation of user interface and are related to every other interface of the system. For instance, input/output devices are quite often influenced by product design. In the course of product design, either the formal or the functional approach dominates, while semiotic considerations (regarding the semiotic unity of the interface) are ignored. User-friendliness — usually more a marketing pitch than a verifiable quality — does not automatically become user-friendliness in terms of physical and mental aspects of working with a system, programming it, or simply using some of its routines. While the problem of the user is central to the design of user interface, the semiotic matrix tries to solve the problem of defining the user (Figure 12). But we have to be more precise and consider all the elements to which the user relates. This can be represented at the system's level (see Figure 13) or at the conceptual level (see Figure 14), in which case it becomes obvious that each user will form his own model when using a given computer. The model developed by each particular user (influenced by manuals, guides, tutorials, etcetera) is the product of

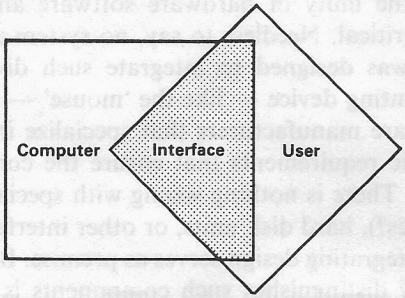

Figure 12. *Semiotic matrix defining the user*

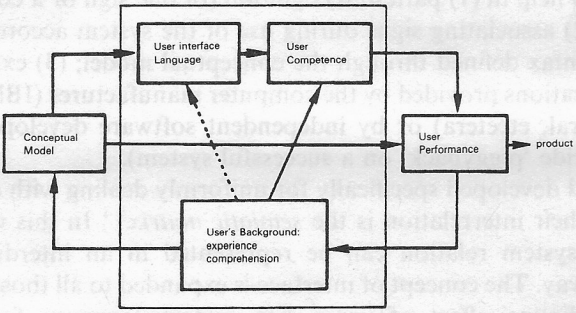

Figure 13. *User related elements at the systems level*

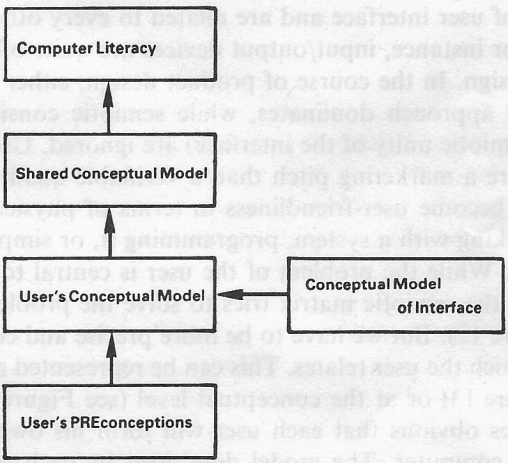

Figure 14. *User related elements at the conceptual level*

'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 each user's model. Preconceptions influence this model; so do other semiotic contexts: cognitive skills, emotional factors, esthetic components, and so on.

Computers are basically used for problem solving, a fact that should be considered carefully when communication issues are approached. As opposed to other tools, the computer is almost a 'universal' problem solver. This means that the tool can be adapted to various tasks through its programs. There is no need to step down from the high-level language of programming to low-level protocols dealing with the concrete problem. Adaptation of the tool to the problem takes place through the *intermediary* of the different interfaces that are part of the system and that simultaneously connect it with outer environments (the user, other systems, communication networks, etcetera). Programs are abstract entities that obey formal rules. Editing a document, for instance, is a concrete activity in which the user causes the abstract entity DOCUMENT, approachable through interface, to have a *concrete reality*: it will receive a name; it will consist of a text; it will be edited/formatted in a particular way, etcetera. In short, using a computer means to make the abstract concrete. Correspondence to the real world (an office, for example) is ensured through the semiotic conventions of the interface and primarily through the conventions of *likeness/resemblance* in the mind of the intended user.

The common representation of *the user* distinguishes the novice from the experienced user. This is a linear representation, very comfortable, but not necessarily appropriate (see Figure 15). 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. A more complex model is necessary, one which at least does not contradict the knowledge the user accumulates while working with various computers. Although experience is important, a semiotic property of computer-aided activity is that in order to understand and use sign systems, a user has to bring into the activity comprehension gained from culture and general education. The improved user model is supposed to help the designer of interface evaluate his choice of signs (see Figure 16). The two-dimensional matrix can be improved, first of all by

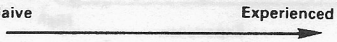

Figure 15. *Simplified user model*

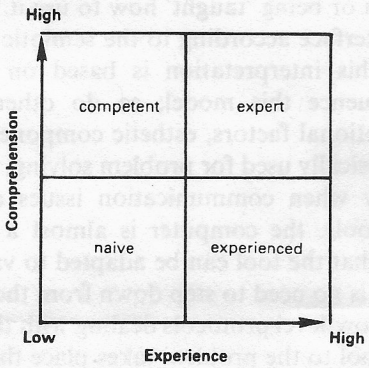

Figure 16. *Improved user model*

involving other qualities that have proved essential in computer use and are acknowledged in computer culture. Imagination, to give just one example, plays an important role in programming and in running programs for some stereotype applications or adapting programs to new functions (see Figure 17). The fuzzy nature of some elements also has to be considered if we indeed intend to deal with a user as concretely determined as possible. An interface that leads to a fast leveling of user performance is the result of unacceptable simplifications in the designer's idea of the future user (the *interpretant* in the complex sign the system represents). Perhaps the matrix should also deal with how we quantify the cognitive modes of human thinking. Whether or not the distinction left-

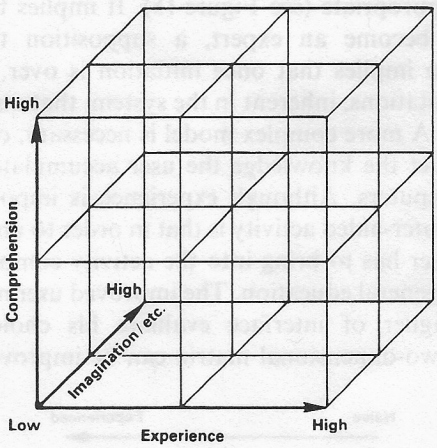

Figure 17. *Suggested user model: a multidimensional matrix*

right hemispheres of the brain can be sustained — 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 (for instance, 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 (Figure 18). Configurational systems of signs are processed 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 processing, integration dominates. Time is related to sequence (our time representations are sequential), while space is related to configuration. 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 homogeneous environment — that is, 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, such as a mouse, some users will rely on emulating keys in order to avoid swift changes that have proven exhausting. 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 19.) Physical properties (of the keyboard, display, printer, pointing device, etcetera) are but an extension

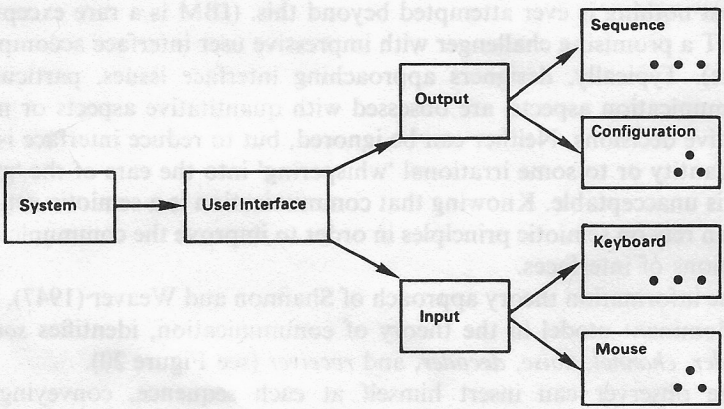

Figure 18. *Sequential processing of symbolic information.* ...=sequence mode; ∴ =configuration mode

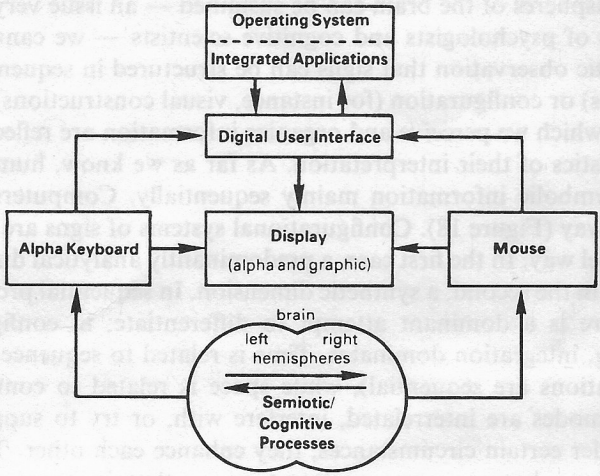

Figure 19.

of the properties of the system in its entirety. While esthetic and functional criteria are difficult to codify, they are part of the interface. Actually, to provide a really user-friendly interface means to make possible not everything, but only what is acceptable. Esthetic and functional acceptability, as well as cultural adequacy, are becoming ever more critical qualities. Only a superficial designer, one who targets the lower level of the market, would think that cultural adequacy is reducible to emulation of characters used in foreign languages. Unfortunately, almost nothing is ever attempted beyond this. (IBM is a rare exception, AT&T a promising challenger with impressive user interface accomplishments). Typically, designers approaching interface issues, particularly communication aspects, are obsessed with quantitative aspects or make intuitive decisions. Neither can be ignored, but to reduce interface issues to quantity or to some irrational 'whispering' into the ears of the 'gifted few' is unacceptable. Knowing that communication is a semiotic activity, we can rely on semiotic principles in order to improve the communication functions of interfaces.

The information theory approach of Shannon and Weaver (1947), long the dominant model in the theory of communication, identifies *source*, *encoder*, *channel*, *noise*, *decoder*, and *receiver* (see Figure 20).

The observer can insert himself at each sequence, conveying his observation in metalanguage. Parameters such as *entropy* or *redundancy* (of the message), while pertinent to 'messages' moved back and forth during computer use, are not the most important and do not begin to

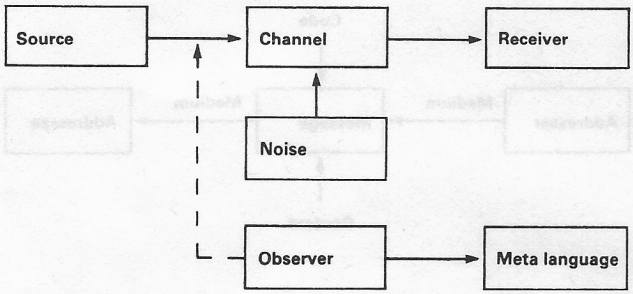

Figure 20. *Information theory approach* (after Shannon and Weaver 1947)

reflect the specific nature of interface activities. Even the semiotically improved model is not adequate, although the role of the repertory of signs used is in evidence (Figure 21). The overlapping of the repertory (attached to the system or the user) is a *necessary* condition for communication, but not a sufficient one. Reasons exist for seeking a more specific model to deal with the sufficient conditions. Such a model should take into account the fact that the communication to take place is not homogeneous, and that mixed systems of signs, as well as different forms of interpretation adapted to these different signs, occur. A first representation, again identifying a source and a receiver, introduces relations that have remained unaccounted for in previous models. Here, the medium (CRT display, printed output, sound, even voice input devices) is important, as are the context and the set of codes applied (from natural language codes, algorithms, and binary codes to codes of high-level programming languages and so on). Figure 23 — inspired mainly by

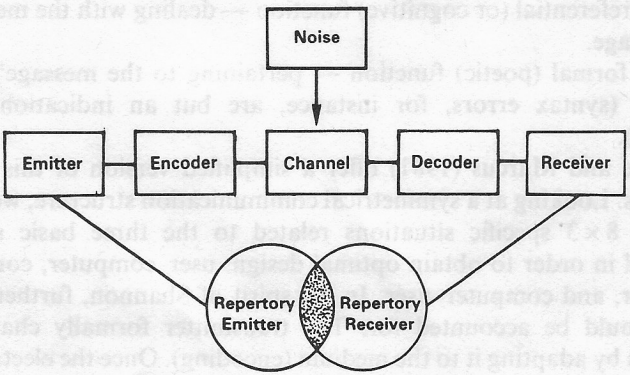

Figure 21. *Semiotic information theory approach*

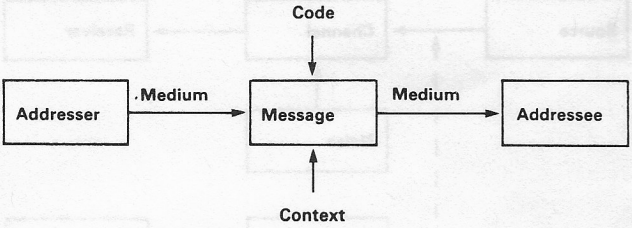

Figure 22.

Bühler's (1933) study of social communication and based on Jakobson's (1960) understanding of linguistic processes — makes possible both an integrative (how the *whole* works) and a differential (part-by-part) approach. Several partial functions can be identified:

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 toward 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 errors, for instance, are but an indication of this function).

Calude and Marcus (1981) offer a simplified version of this table of functions. Looking at a symmetrical communication structure, we have to consider 8×3 specific situations related to the three basic segments identified in order to obtain optimal design: user–computer, computer–computer, and computer–user. In the spirit of Shannon, further distinctions should be accounted for. The transmitter formally changes the messages by adapting it to the medium (encoding). Once the electric signal arrives at the intended destination, decoding — that is, reconstitution of the message in a form adequate to the computer layer addressed

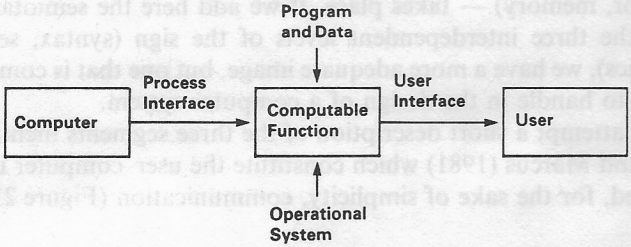

Figure 24.

proach 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 (see Zadeh 1984). Looking from the computer to the user, we see a slightly different picture (Figure 24). 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 computable function step by step (one thing at a time); that is, 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 1987) 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 (Figure 25). Very relevant here is the metalinguistic function, since what actually goes on is 'transition' (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

(processor, memory) — takes place. If we add here the semiotic distinction of the three interdependent levels of the sign (syntax, semantics, pragmatics), we have a more adequate image, but one that is complex and not easy to handle in the design of a computer system.

Let us attempt a short description of the three segments mentioned by Calude and Marcus (1981) which constitute the user–computer interrelation called, for the sake of simplicity, communication (Figure 23).

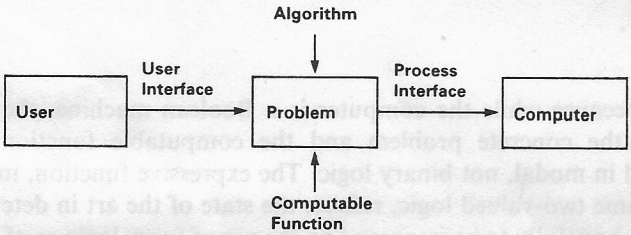

Figure 23. *The user-computer interrelation*

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 computer crashing?) and involves the relations among user–computable function, computable function–program, program–computer, and 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. Since each system comes with specifications impossible to avoid, whether or not the given system can accept the program becomes an issue; and if it can, how effective is the program going to be? 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 types of problems a system is supposed to assist the user in solving. Two different forms of 'intelligence' are evident: the 'intelligence' built (wired) into 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), etcetera. The referential function is difficult to ap-

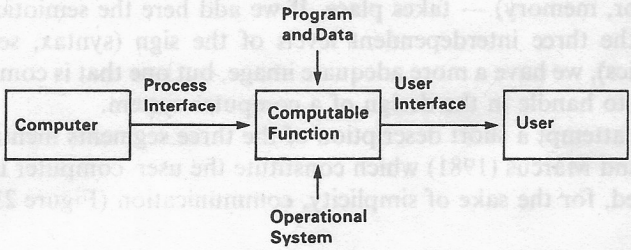

Figure 24.

proach 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 (see Zadeh 1984). Looking from the computer to the user, we see a slightly different picture (Figure 24). 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 computable function step by step (one thing at a time); that is, 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 1987) 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 (Figure 25). Very relevant here is the metalinguistic function, since what actually goes on is 'transition' (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

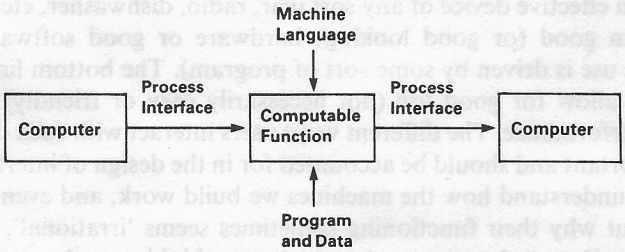

Figure 25.

takes place and, in the case of more advanced systems, the so-called level of gravity (permissiveness of the system) as well. Some languages support this function better 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 26). 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. For reasons difficult to understand, interface designers treat application environments and programming and debugging environments as though they were totally independent. Basically, this treatment makes several distinct channels of communication necessary, a decision that deserves further examination from the hardware and software perspectives. This issue brings me to the last aspect of the semiotics of computer interface.

Everybody knows, or would agree, that an effective computer, and in

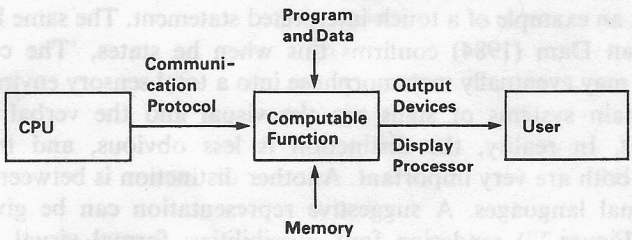

Figure 26.

general an effective device of any sort (car, radio, dishwasher, etcetera), is more than good (or good looking) hardware or good software (each device we use is driven by some sort of program). The bottom line is that it should allow for good use (not necessarily easy or friendly) and for quality performance. The different ways users interact with such devices is very important and should be accounted for in the design of interface. 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. As already mentioned, interfacing goes well beyond these aspects and extends to everything a user will come into contact with when using a system and getting output from it (on CRT display, slide, film, hard copy, etcetera). 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'.[5]
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. Interface is a semiotic issue on both of the levels at which it takes place: process interface and user interface. Our expectations are reliability (with tolerance toward 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 languages, as many senses as we have. A 'taste' statement can go as a mixture of or succession from sour, bitter, sweet. ... Hot, warm cold ... can represent an example of a touch-interpreted statement. The same holds for smell. Van Dam (1984) confirms this when he states, 'The computer interface may eventually metamorphose into a total sensory environment'.

The main systems 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 (Figure 27) rendering four possibilities: formal–visual, formal–verbal, natural–visual, and natural–verbal.

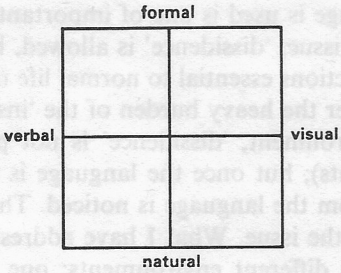

Figure 27. *Language matrix.*

Voice input devices, or other I/O features (heat sensitive or 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 visual–verbal distinction refers first of all to possible forms of representation. The natural–formal distinction refers to the logical structure and thus to the nature of language (cultural versus artificial). The matrix takes combinations into consideration too. Formal verbal language may prove difficult for people to read or write. Such languages require special training and a competence level demanded by specialized fields (mathematics, symbolic logic, language programming, dance, music, etcetera). Reading texts written in a programming language, or a musical score, or a choreographic labanotation is difficult. Formal visual languages may prove difficult to 'write' but can be read more easily (though 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 (for example, diagrams, charts, lists, etctera). Although controlled by grammar, natural languages are easier to write because this grammar is not as rigidly specified as the grammar of formal languages. On the other hand, it is harder to be specific and precise, 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) maintains 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). If we again use the metaphor of 'governing', the environment in

which natural language is used is one of important amounts of 'governing'. Type is not the issue; 'dissidence' is allowed, but the regime is very bureaucratic. Transactions essential to normal life involve 'red tape'. The language user is under the heavy burden of the 'institution of language'. In the artificial environment, 'dissidence' is not possible (especially in compiler environments); but once the language is used according to its rules, no pressure from the language is noticed. The amount of 'governing' is small; type is the issue. What I have addressed here are issues of language use in two different environments: one in which the user is comfortable, since it is the environment of his everyday life; and another in which the user faces something less familiar, in which interface should play a mediating role. 'The ideal situation', as Van Dam (1984) describes 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'. This should be contradicted, not only because it raises false hopes, but also because the underlying principles of digital computers are those of Boolean logic — a reduction from the multivalued logic of natural languages to two-valued logic. 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. Interface is a trade-off in which amount and type (of signs used) are the fundamental parameters. Norman (1983), who introduced a remarkable quantitative method for trade-off analysis, makes the basic statement: 'Any single design technique is apt to have its virtues along one dimension compensated by deficiencies along another'. Maybe the following comparative charts (Table 2) will explain the kind of trade-off implicit in the semiotic decision

Table 2. *Language chart.*

Natural language	Formal language	Communications characteristics of formal languages		
		write	*select*	*read*
general	specific	understand	read system	model system
unlimited	limited vocabulary	system	intuition	intuition
vocabulary	predefined grammar	concrete	have purpose	indefinite
indefinite grammar	logical structure	knowledge	semantics	purpose
intuitive structure	difficult to acquire	have purpose	error	semantics
easily acquired	competence	syntax error		error
competence	easy to attain high			
difficult to obtain	performance/easy			
performance	to learn			

involved in the design of interfaces. An integrated, interdisciplinary approach to interface considers the contribution of each component. Semiotics coordinates the relationship between everything that participates in interfacing. Product design, software engineering, hardware, ergonomics, etcetera — highly specialized fields — should each be evaluated in turn and integrated in the comprehensive language of the product. Of course, semiotics has to provide the necessary means required. This article has given some examples.

Notes

* 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 (Spring 1984) during his tenure as William A. Kern Institute Professor in Communications and 'Computer Graphics–Graphic Design Issues', Autumn 1984, RISD.

1. Registered trademark, Apple Computer Corporation.
2. Registered trademark, Xerox, Inc.
3. Copyright Mihai Nadin, 1981.
4. Registered trademark, Nadin and Ockerse, Ltd., 1983.
5. Source unknown.

References

Bar Hillel, Yehoshua (1970). Communication and argumentation in pragmatic languages. In *Linguagi nella Società nella Tecnica*, 129–168. Milan: Covegno Olivetti.

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

Calude, Cristian and Marcus, Soloman (1981). Introduction to the semiotics of man — computer communication. *Revue Roumaine de Linguistique* 31, 191–211.

Jakobson, Roman (1960). Linguistics and poetics. In *Style and Language*, T. A. Sebeok (ed.), 350–377. Cambridge: The MIT Press.

Meyrowitz, Norman and Van Dam, Andries (1982). Interactive editing systems. *Computing Surveys* 14 (3), 323.

Molzberger, Peter (1983). *Aesthetics and Programming CHI '83*, Proceedings. New York: Association for Computing Machinery.

Morris, Charles (1938). *Foundations of the Theory of Signs*. Chicago: University of Chicago Press.

Nadin, Mihai (1981). *Zeichen und Wert*. Tübingen: Narr Verlag.

Norman, D. A. (1983). Design principles for human–computer interfaces. In *Human Factors in Computing Systems CHI '83*, Proceedings, vol. 1. New York: ACM.

Peirce, Charles Sanders (1931–1966). *The Collected Papers of Charles Sanders Peirce*, C. Hartshorne, P. Weiss, and A. W. Burks (eds.). Cambridge, MA: Harvard University Press.

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

Schneider, M. L. and Thomas, J. C. (eds.) (1983). The humanization of computer interfaces (Introduction). *Communications of the Association for Computing Machinery*. 26 (4), 252–253.

Shannon, Claude and Weaver, Warren (1947). *The Mathematical Theory of Communication*. Urbana: University of Illinois Press.

Simon, Herbert (1982). *The Sciences of the Artificial*. Cambridge, MA: The MIT Press.

van Dam, Andries (1984). Computer graphics comes of age. *Communications of the Association for Computing Machinery*. 27 (7), 646.

Zadeh, Lofti (1984). Coping with the impression of the real world. *Communications of the Association for Computing Machinery*. 27, 304–311.

Mihai Nadin is Eminent Scholar, Professor in Art and Design Technology at The Ohio State University. From 1980–1985, he was Professor in the Liberal Arts and Graphic Design Divisions at the Rhode Island School of Design and director of the Institute for Visual Communication and Semiotics. His principal research interests are foundations of semiotics, the semiotics of computer-aided human activity, art and design issues in artificial intelligence, and fuzzy logic/vagueness, on which he has written extensively. He lectures at several universities in the USA and Europe and is a consultant in visual and computer-aided communication.