User Interface: A Personal View

Alan Kay
Apple Fellow
Apple Computer, Inc.

WHEN I WAS ASKED to write this chapter, my first reaction was "A book on user interface design—does that mean it's now a real subject?" Well, as of 1989, it's still yes and no. User interface has certainly been a hot topic for discussion since the advent of the Macintosh. Everyone seems to want user interface but they are not sure whether they should order it by the yard or by the ton. Many are just now discovering that user interface is not a sandwich spread—applying the Macintosh style to poorly designed applications and machines is like trying to put Béarnaise sauce on a hotdog!

Of course the practice of user interface design has been around at least since humans invented tools. The unknown designer who first put a haft on a hand axe was trying not just to increase leverage but also to make it an extension of the *arm*, not just the fist. The evolutionary designer whom Richard Dawkins calls the Blind Watchmaker [Dawkins, 1986] has been at it much longer; all of life's startling interfitness is the result. A more recent byproduct of the industrial revolution called *ergonomics* in Europe and *human factors* in the U.S. has studied how the human body uses senses and limbs to work with tools. From the earliest use of interactive computing in the fifties—mostly for air traffic control and defense—there have been attempts at user interface design and application of ergonomic principles. Many familiar components of modern user interface design appeared in the fifties and early sixties, including pointing devices, windows, menus, icons, gesture recognition, hypermedia, the first personal computer, and

more. There was even a beautifully designed user interface for an end-user system in JOSS—but its significance was appreciated only by its designer and users.

Therefore, let me argue that the actual dawn of user interface design first happened when computer designers finally noticed, not just that end users had functioning minds, but that a better understanding of how those minds worked would completely shift the paradigm of interaction.

This enormous change in point of view happened to many computerists in the late sixties, especially in the ARPA research community. Everyone had their own catalyst. For me it was the FLEX machine, an early desktop personal computer of the late sixties designed by Ed Cheadle and myself.

Based on much previous work by others, it had a tablet as a pointing device, a high-resolution display for text and animated graphics, and multiple windows, and it directly executed a high-level object-oriented end-user simulation language. And of course it had a "user interface," but one that repelled end users instead of drawing them closer to the hearth. I recently revisited the FLEX machine design and was surprised to find how modern its components were—even a use of icon-like structures to access previous work.

But the combination of ingredients didn't gel. It was like trying to bake a pie from random ingredients in a kitchen: baloney instead of apples, ground-up Cheerios instead of flour, etc.

Then, starting in the summer of 1968, I got hit on the head randomly but repeatedly by some really nifty work. The first was just a little piece of glass at the University of Illinois. But the glass had tiny glowing dots that showed text characters. It was the first flat-screen display. I and several other grad students wondered when the surface could become large and inexpensive enough to be a useful display. We also wondered when the FLEX machine silicon could become small enough to fit on the back of the display. The answer to both seemed to be the late seventies or early eighties. Then we could all have an inexpensive powerful notebook computer—I called it a "personal computer" then, but I was thinking *intimacy*.

I read McLuhan's *Understanding Media* [1964] and understood that the most important thing about any communications medium is that message receipt is really message recovery; anyone who wishes to receive a message embedded in a medium must first have internalized the medium so it can be "subtracted" out to leave the message behind. When he said "the medium is the message" he meant that you have to *become* the medium if you use it.

That's pretty scary. It means that even though humans are the animals that shape tools, it is in the nature of tools and man that learning to use tools reshapes us. So the "message" of the printed book is, first, its availability to individuals, hence, its potential detachment from extant social

processes; second, the uniformity, even coldness of noniconic type, which detaches readers from the vividness of the now and the slavery of commonsense thought to propel them into a far more abstract realm in which ideas that don't have easy visualizations can be treated.

McLuhan's claim that the printing press was the dominant force that transformed the hermeneutic Middle Ages into our scientific society should not be taken too lightly—especially because the main point is that the press didn't do it just by making books more available, it did it by changing the thought patterns of those who learned to read.

Though much of what McLuhan wrote was obscure and arguable, the sum total to me was a shock that reverberates even now. The computer is a medium! I had always thought of it as a tool, perhaps a vehicle—a much weaker conception. What McLuhan was saying is that if the personal computer is a truly new medium then the very use of it would actually change the thought patterns of an entire civilization. He had certainly been right about the effects of the electronic stained-glass window that was television—a remedievalizing tribal influence at best. The intensely interactive and involving nature of the personal computer seemed an antiparticle that could annihilate the passive boredom invoked by television. But it also promised to surpass the book to bring about a new kind of renaissance by going beyond static representations to dynamic simulation. What kind of a thinker would you become if you grew up with an active simulator connected, not just to one point of view, but to all the points of view of the ages represented so they could be dynamically tried out and compared? I named the notebook-sized computer idea the Dynabook to capture McLuhan's metaphor in the silicon to come.

Shortly after reading McLuhan, I visited Wally Feurzeig, Seymour Papert, and Cynthia Solomon at one of the earliest LOGO tests within a school. I was amazed to see children writing programs (often recursive) that generated poetry, created arithmetic environments, and translated English into Pig Latin. And they were just starting to work with the new wastepaper-basket-sized turtle that roamed over sheets of butcher paper making drawings with its pen.

I was possessed by the analogy between print literacy and LOGO. While designing the FLEX machine I had believed that end users needed to be able to program before the computer could become truly theirs—but here was a real demonstration, and with children! The ability to "read" a medium means you can *access* materials and tools created by others. The ability to "write" in a medium means you can *generate* materials and tools for others. You must have both to be literate. In print writing, the tools you generate are rhetorical; they demonstrate and convince. In computer writing, the tools you generate are processes; they simulate and decide.

If the computer is only a vehicle, perhaps you can wait until high school

to give "driver's ed" on it—but if it's a medium, then it must be extended all the way into the world of the child. How to do it? Of course it has to be done on the intimate notebook-sized Dynabook! But how would anyone "read" the Dynabook, let alone "write" on it?

LOGO showed that a special language designed with the end user's characteristics in mind could be more successful than a random hack. How had Papert learned about the nature of children's thought? From Jean Piaget, the doyen of European cognitive psychologists [see Piaget, 1926, 1928, and 1952]. One of his most important contributions is the idea that children go through several distinctive intellectual stages as they develop from birth to maturity. Much can be accomplished if the nature of the stages is heeded and much grief to the child can be caused if the stages are ignored. Piaget noticed a kinesthetic stage, a visual stage, and a symbolic stage. An example is that children in the visual stage, when shown a squat glass of water poured into a tall thin one, will say there is more water in the tall thin one even though the pouring was done in front of their eyes.

One of the ways Papert used Piaget's ideas was to realize that young children are not well equiped to do "standard" symbolic mathematics until the age of 11 or 12, but that even very young children can do other kinds of math, even advanced math such as topology and differential geometry, when it is presented in a form that is well matched to their current thinking processes. The LOGO turtle with its local coordinate system (like the child, it is always at the center of its universe) became a highly successful "microworld" for exploring ideas in differential geometry.

This approach made a big impression on me and got me to read many more psychology books. Most (including Piaget's) were not very useful, but then I discovered Jerome Bruner's *Towards a Theory of Instruction* [1966]. He had repeated and verified many of Piaget's results, and in the process came up with a different and much more powerful way to interpret what Piaget had seen. For example, in the water-pouring experiment, after the child asserted there was more water in the tall thin glass, Bruner covered it up with a card and asked again. This time the child said, "There must be the same because where would the water go?" When Bruner took away the card to again reveal the tall thin glass, the child immediately changed back to saying there was more water.

When the cardboard was again interposed the child changed yet again. It was as though one set of processes was doing the reasoning when the child could see the situation, and another set was employed when the child could not see. Bruner's interpretation of experiments like these is one of the most important foundations for human-related design. Our mentalium seems to be made up of multiple separate mentalities with very different characteristics. They reason differently, have different skills, and often are in conflict. Bruner identified a separate mentality with each of Piaget's

stages: he called them *enactive, iconic, symbolic*. While not ignoring the existence of other mentalities, he concentrated on these three to come up with what are still some of the strongest ideas for creating learning-rich environments.

The work of Papert convinced me that whatever user interface design might be, it was solidly intertwined with learning. Bruner convinced me that learning takes place best environmentally and roughly in stage order—it is best to learn something kinesthetically, then iconically, and finally the intuitive knowledge will be in place that will allow the more powerful but less vivid symbolic processes to work at their strongest. This led me over the years to the pioneers of environmental learning: Montessori Method, Suzuki Violin [Suzuki, 1969], and Tim Gallwey's *Inner Game of Tennis* [1974], to name just a few.

My point here is that as soon as I was ready to look deeply at the human element, and especially after being convinced that the heart of the matter lay with Bruner's multiple mentality model, I found the knowledge landscape positively festooned with already accomplished useful work. It was like the man in Moliere's *Bourgeois Gentilhomme* who discovered that all his life he had been speaking prose! I suddenly remembered McLuhan: "I don't know who discovered water but it wasn't a fish." Because it is in part the duty of consciousness to represent ourselves to ourselves as simply as possible, we should sorely distrust our commonsense self view. It is likely that this mirrors-within-mirrors problem in which we run into a misleading commonsense notion about ourselves at every turn is what forced psychology to be one of the most recent sciences—if indeed it yet is.

Now, if we agree with the evidence that the human cognitive facilities are made up of a *doing* mentality, an *image* mentality, and a *symbolic* mentality, then any user interface we construct should at least cater to the mechanisms that seem to be there. But how? One approach is to realize that no single mentality offers a complete answer to the entire range of thinking and problem solving. User interface design should integrate them at least as well as Bruner did in his spiral curriculum ideas.

One of the implications of the Piaget-Bruner decomposition is that the mentalities originated at very different evolutionary times and there is little probability that they can intercommunicate and synergize in more than the most rudimentary fashion. In fact, the mentalities are more likely to interfere with each other as they compete for control. The study by Hadamard on math and science creativity [1945] and others on music and the arts indicate strongly that creativity in these areas is not at all linked to the symbolic mentality (as most theories of teaching suppose), but that the important work in creative areas is done in the the initial two mentalities—most in the iconic (or figurative) and quite a bit in the enactive. The groundbreaking work by Tim Gallwey on the "inner game" [1974] showed

what could be done if interference were removed (mentalities not relevant to the learning were distracted) and attention was facilitated (the mentalities that could actually do the learning were focused more strongly on the environment).

Finally, in the 60s a number of studies showed just how modeful was a mentality that had "seized control"—particularly the analytical-problem-solving one (which identifies most strongly with the Bruner symbolic mentality). For example, after working on five analytic tasks in a row, if a problem was given that was trivial to solve figuratively, the solver could be blocked for hours trying to solve it symbolically. This makes quite a bit of sense when you consider that the main jobs of the three mentalities are:

enactive	know where you are, manipulate
iconic	recognize, compare, configure, concrete
symbolic	tie together long chains of reasoning, abstract

The visual system's main job is to be interested in everything in a scene, to dart over it as one does with a bulletin board, to change context. The symbolic system's main job is to stay with a context and to make indirect connections. Imagine what it would be like if it were reversed. If the visual system looked at the object it first saw in the morning for five hours straight! Or if the symbolic system couldn't hold a thought for more than a few seconds at a time!

It is easy to see that one of the main reasons that the figurative system is so creative is that it tends not to get blocked because of the constant flitting and darting. The chance of finding an interesting pattern is very high. It is not surprising, either, that many people who are "figurative" have extreme difficulty getting anything finished—there is always something new and interesting that pops up to distract. Conversely, the "symbolic" person is good at getting things done, because of the long focus on single contexts, but has a hard time being creative, or even being a good problem solver, because of the extreme tendency to get blocked. In other words, because none of the mentalities is supremely useful to the exclusion of the others, the best strategy would be to try to gently force synergy between them in the user interface design.

Out of all this came the main slogan I coined to express this goal:

Doing with Images makes Symbols

The slogan also implies—as did Bruner—that one should start with—be grounded in—the concrete "Doing with Images," and be carried into the more abstract "makes Symbols."

All the ingredients were already around. We were ready to notice what the theoretrical frameworks from other fields of Bruner, Gallwey, and others were trying to tell us. What is surprising to me is just how long it took to

put it all together. After Xerox PARC provided the opportunity to turn these ideas into reality, it still took our group about five years and experiments with hundreds of users to come up with the first practical design that was in accord with Bruner's model and really worked.

DOING	mouse	*enactive*	know where you are, manipulate
with			
IMAGES	icons, windows	*iconic*	recognize, compare, configure, concrete
makes			
SYMBOLS	Smalltalk	*symbolic*	tie together long chains of reasoning, abstract

Part of the reason perhaps was that the theory was much better at confirming that an idea was good than at actually generating the ideas. In fact, in certain areas like "iconic programming," it actually held back progress, for example, the simple use of icons as signs, because the siren's song of trying to do symbolic thinking iconically was just too strong.

Some of the smaller areas were obvious and found their place in the framework immediately. Probably the most intuitive was the idea of multiple overlapping windows. NLS had multiple panes, FLEX had multiple windows, and the bit-map display that we thought was too small, but that was made from individual pixels, led quickly to the idea that windows could appear to overlap. The contrastive ideas of Bruner suggested that there should always be a way to compare. The flitting-about nature of the iconic mentality suggested that having as many resources showing on the screen as possible would be a good way to encourage creativity and problem solving and prevent blockage. An *intuitive* way to use the windows was to activate the window that the mouse was in and bring it to the "top." This interaction was *modeless* in a special sense of the word. The active window constituted a mode to be sure—one window might hold a painting kit, another might hold text—but one could get to the next window to do something in *without any special termination*. This is what *modeless* came to mean for me—the user could always get to the next thing desired without any backing out. The contrast of the nice modeless interactions of windows with the clumsy command syntax of most previous systems directly suggested that everything should be made modeless. Thus began a campaign to "get rid of modes."

The object-oriented nature of Smalltalk was very suggestive. For example, *object-oriented* means that the object knows what it can do. In the abstract symbolic arena, it means we should first write the object's name (or whatever will fetch it) and then follow with a message it can understand that asks it to do something. In the concrete user-interface arena, it suggests that we should select the object first. It can then furnish us with a menu of

what it is willing to do. In both cases we have the *object* first and the *desire* second. This unifies the concrete with the abstract in a highly satisfying way.

The most difficult area to get to be modeless was a very tiny one, that of elementary text editing. How to get rid of "insert" and "replace" modes that had plagued a decade of editors? Several people arrived at the solution simultaneously. My route came as the result of several beginning programmer adults who were having trouble building a paragraph editor in Smalltalk, a problem I thought should be easy. Over a weekend I built a sample paragraph editor whose main simplification was that it eliminated the distinction between insert, replace, and delete by allowing selections to extend *between* the characters. Thus, there could be a zero-width selection, and thus every operation could be a replace. "Insert" meant replace the zero-width selection. "Delete" meant replace the selection with a zero-width string of characters. I got the tiny one-page program running in Smalltalk and came in crowing over the victory. Larry Tesler thought it was great and showed me the idea, already working in his new Gypsy editor (which he implemented on the basis of a suggestion from Peter Deutsch). So much for creativity and invention when ideas are in the air. As Goethe noted, the most important thing is to enjoy the thrill of discovery rather than to make vain attempts to claim priority!

Quo Userus Interfacus?

Or, now that the Mac way of doing things has taken hold, will we ever be able to get rid of it? If the IBM 3270/PC way of doing things is "machine code," doesn't that make the Mac the COBOL of user interface designs?

One way to look at the prospects for improvement in the future is to consider user interface as just another ripple in the main current of human extension over the past several hundred thousand years. As I see it, we humans have extended ourselves in two main ways. The first to leap to anyone's mind is the creation of *tools*—physical tools like hammers and wheels and figurative mental tools like language and mathematics. To me, these are all extensions of gesture and grasp. Even mathematics, which is sometimes thought of as forbiddingly vague, is actually a way to take notions that are too abstract for our senses and make them into little symbols that we can move around. The M-word for tools is *manipulation*. The second way we have extended ourselves is more subtle. Lewis Mumford points out that for most of our history the most complicated machines constructed by humans have mostly had humans as moving parts [Mumford, 1934 and 1967–70]. In other words, we are creatures who both like to convince others to work on our goals and are willing to be convinced in turn to work on the goals of others. Mumford calls these structures megamachines; I call the process goal cloning. The M-word here is *management*. We manipulate tools but manage people.

Both of these ideas were applied early to computer-user interface designs. So far in this chapter we have been discussing tool-based ideas, because the Macintosh is the most well known of all tool-based interfaces. Yes, there is still considerable life in the computer-based tool that is manipulated, but it is starting to drown in incremental "enhancements." It took Hercules a considerable time to clean the Augean stables, and it may take just as long to get the next real step in tool-based design. As always, the key is to forget the vividness of the present and to get back to basic principles that are centered directly on the human condition. As is often the case, it is not how well we can learn but how well we can *unlearn* that will make the difference. We have to discard fondly held notions that have sneakily become myths while we weren't watching. An old adage is that no biological organism can live in its own waste products—and that seems just as true for computer designs. If things continue as they are now going, we will soon be able to see the original Lisa and Mac as an improvement on their successors!

One of the most compelling snares is the use of the term *metaphor* to describe a correspondence between what the users see on the screen and how they should think about what they are manipulating. My main complaint is that *metaphor* is a poor metaphor for what needs to be done. At PARC we coined the phrase *user illusion* to describe what we were about when designing user interface. There are clear connotations to the stage, theatrics, and magic—all of which give much stronger hints as to the direction to be followed. For example, the screen as "paper to be marked on" is a metaphor that suggests pencils, brushes, and typewriting. Fine, as far as it goes. But it is the magic—understandable magic—that really counts. Should we transfer the paper metaphor so perfectly that the screen is as hard as paper to erase and change? Clearly not. If it is to be like magical paper, then it is the *magical* part that is all important and that must be most strongly attended to in the user interface design.

While the magic is being designed, the very idea of a paper "metaphor" should be scrutinized mercilessly. One of the most wonderful properties of a computer is that no matter how many dimensions one's information has, a computer representation can always supply at least one more. One result is that any seeming distance between items in our world of limited dimension can be competely "disappeared."

This is something that Vannevar Bush and his chief prophet Doug Englebart noticed immediately, and hypermedia was born. In a world of Dynabooks, information will not be printed—it would destroy most of the useful associations—and something much more than superpaper will emerge. The notion of hypermedia is much more a "user illusion" than a "metaphor."

Let me attack a few more sacred cows. For example, the desktop "metaphor." I don't want a screen that is much like my physical desk. It just

gets messy—yet I hate to clean up while in the middle of a project. And I'm usually working on lots of different projects at the same time. At PARC they used to accuse me of filling up a desk until it was uselessly tangled and then abandoning it for another one! One solution to this is "project views" as originally implemented by Dan Ingalls in Smalltalk-76. Again, this is more of a user illusion than a metaphor. Each area holds all the tools and materials for a particular project and is automatically suspended when you leave it. A bit like multiple desks—but it is the magic that is truly important. Here the magic is that every change to the system that is made while in a project area—no matter how deeply made to the system (even changing the meaning of integer "+"!)—is logged locally to the project area. Yet each of the project areas are parallel processes that can communicate with one another. This is a user illusion that is easy to understand yet doesn't map well to the physical world.

How about the folder? One of my longstanding pet hates is to have them behave anything like their physical counterparts. For example, as they existed in Officetalk, Star, Lisa, and Mac—like real folders—there is only one icon for a document or application and it can be in only one folder. This drives me crazy, because the probability of *not* finding what you are looking for by browsing has just been maximized! It is trivial to have as many icon instances for a given doc or app in as many folders as one wishes. They should be near any place where they might be useful. (Dragging a singleton out on the desktop is not a solution to this problem!) But even if that were fixed we have to ask: why a folder? Instead of passive containers, why not have active retrievers that are constantly trying to capture icon instances that are relevant to them? Let's call them *bins*. Imagine having a "memos" bin that, whenever and wherever you make up a memo, captures a copy of the doc icon. You might have a "memos to boss" bin that automatically captures only those icons of docs sent to your boss. Folders kill browsing. Bins extend its useful range.

That wonderful system, HyperCard, in spite of its great ideas, has some "metaphors" that set my teeth on edge. Four of them are "stack," "card," "field," and "button." In "stack" we find grievously unnecessary limitations, not the least of which is the strange notion that only one stack can be in front of us at a time.[1] This is not just imitating paper with a vengeance—it is building in a limitation not imposed by the physical world. Thus, when we need to get "help" we are forced away from the very image we need help with in order to delve into the help stack—and this on the very machine that first successfully introduced multiple windows to the commercial world!

Again, to echo a previous complaint about folders and icons, we can't

[1] This is no longer true in HyperCard version 2.0.

directly have a card in more than one stack—this is partly because stacks serve some of the purpose of a class in an object-oriented language. However, this confounds the use of a stack as a set or collection of useful items. Being able to copy a card into a new stack doesn't count, because it is dynamic fidelity we are interested in most of the time.

Finally, in spite of their obvious similarities, stacks, cards, buttons, fields, and values are all quite separate entities. So a card can't contain a stack. A variable can't contain a stack, card, field, or button (or anything but a string value). You can paint on a card but you can't paint on a button. You can make a button or field a different size and move it around but you can't do this with a card. And so forth.

Here the metaphors have almost completely sunk the magic.

Why not just have one kind of object that can contain objects? The objects can be contained sequentially (relative to their before-after objects) or randomly (relative to their containing object). All the objects are sensitive to the mouse and messages, and are scripted. "Stacks" and "fields" are just objects as sequential containers. "Cards" are objects as random containers, themselves contained in a sequential "stack" container. "Buttons" are just small painted cards. And so on. Often design is best done not by adding features but by consolidating them into a more unitary concept.

Perhaps even more frustrating is the metaphor as a siren's song. The one that has led me the farthest without satisfying results has been the attempt to use icons for generative literacy—end-user programming. By the late sixties, the use of icons as signs in computer graphics had been in practice for years. Informally, they seemed to work much better than labels. Semiotics provided one explanation. In semiotic theory a sign is not a word but actually constitutes an entire sentence. Thus, to the extent that the sign could be recognized, it was a very efficient way of saying something. The work by Haber [1970] suggested that it was easy for humans to learn iconic signs and that the image mentality was efficient in finding a given one from a bulletin-board-like collection. Several interactive systems were built using this idea. An early one in Smalltalk was the Shazam animation system in which all interaction was controlled by icons.

The difficulty—perhaps the snare—of icons lies in trying to go beyond their obvious efficiency as signifiers. I was probably the worst offender because I believed then (and still do) that the nirvana of personal computing comes when the end users can change their tools and build new ones without having to become professional-level programmers. I believed then (and still do) that the key is to find a context in which most of the things they want to do are as obvious as, say, moving furniture around in a house. And much of the key has to be iconic because it is the only system in which the end user doesn't have to remember dozens of facts and processes.

Almost everything to the iconic mentality is "before-after," like a bird

building a nest. The current state of things suggests what to do next. Extensive top-down planning is not required—just squish things around until you like the total effect. Of course, in a playpen like that, plans do come to mind and can be useful. But major things have to be accomplished by bottom-up exploration with obvious ingredients or end-user programming will not come to pass.

Well, I still believe all this. The question is whether it can all be done iconically. Is it possible to perform one of those nifty Brunerian feats in which an entire cognitive area—the symbolic—is shifted to another—the iconic—to permit long chains of abstract construction to be carried out in a concrete world? I certainly thought so, and I have to admit still feel a great yearning as I write down the old dream. All I can say is that we and others came up with many interesting approaches over the years but none have successfully crossed the threshold to the end user.

One of the problems is how to get concrete signs to be more abstract without simply evoking the kinds of symbols used by the symbolic mentality. More difficult is how to introduce context in a domain whose great trait is its modeless context-freeness. In most iconic languages, it is much easier to write the patterns than it is to read them. One of the most interesting puzzles in iconic programming (and iconic communication in general) is why there is such a disparity between how understandable images are while they are actively being constructed and how obscure even one's own constructions can appear even the next day. I believe that the major reasons for this can be accounted for as a consequence of *semantic focus*—analogous to that of foveal vision, but having to do with the amount of meaning and connectivity that can be solved by looking at a diagram. A suggestive explanation is found in Minsky's and Papert's book *Perceptrons* [1969], where they show that certain diagrams are beyond the level of complexity that a perceptron of a given aperture can solve.

It is likely that our biological structures have similar limitations at the iconic syntax level. Two other factors also are at work. The first is a "semantic fovea"—an area limitation of content understanding quite analogous to the "perceptron" limitations having to do with simple connectivity. The second factor is a property of images in general—their unsortedness. In other words, unlike paragraphs and lists of words, images have no *a priori* order in which they should be understood. This means that someone coming onto an image from the outside has no strategy for solving it. Painters throughout history have built partial solutions into the paintings themselves, by organizing the compositions to move the viewer's eye about without their being consciously aware of the movement.

One learns strategies for certain types of images. In circuit diagram reading, for example, one often starts with the power supply or with certain obvious "subdiagrams" (such as an amplifier) and starts to ask and answer questions about the surrounding components.

It is obvious why constructing a diagram is much less difficult than trying to read one. During construction, one is controlling focus by looking only at the parts that seem relevant at the time. The general strategy of the diagram is already part of the context and doesn't have to be ferreted out. The analogy to writing prose is quite clear, but prose has the advantage that a writer can employ considerable foreshadowing and building up of context before getting to the main point. The scene can be set at will. Except for a few stereotypes, this is extremely difficult to do with an image. What does this imply for iconic communication?

First, that there are severe limitations on the complexity of the image that can be solved by the visual system. An image of any complexity must therefore be one that has already been learned—essentially as a single symbol—regardless of the apparent complexity of its parts. An image that consists of two or more already learned images cannot have much more than "an association" with its sibling images. Connections between the subimages are likely not to be interpreted correctly. It would not be surprising if the visual system were less able in this area than the mechanisms that solve noun phrases for natural language.

Although it is not fair to say that "iconic languages can't work" just because no one has been able to design a good one, it is likely that the above explanation is close to the truth. If so, a major new direction that should be taken is toward a *rhetoric* for programs expressed as something more like essays—or more accurately, in the hypermedia equivalent of an essay. In the TEX books, Knuth has furnished us with several examples of a master programmer and writer in conjunction with WEB, a rhetorical program formatter, to produce outstandingly readable programs. The challenge would be to produce a language in which the *act* of programming produces within it an understandable explanation.

There is surely much more to be said about tool-based interfaces—it is clear that the few small ideas in the Mac have scarcely scratched the range of expression in our hundred-thousand-year odyssey of tool design. *Manipulation* is still vibrantly alive, not exhausted. But it is now time to consider *management* of intelligent computer processes as an inevitable partner to tool-based work and play. I trace the modern origin of this idea back to John McCarthy's *Advice Taker* of the late fifties, in which he points out that we will soon be living inside vast networked "information utilities" and will require an intelligent assistant to find resources useful to us. A retrieval "tool" won't do because no one wants to spend hours looking through hundreds of networks with trillions of potentially useful items. This is a job for intelligent background processes that can successfully clone their users' goals and then carry them out. Here we want to *indirectly manage agents,* not directly manipulate objects.

The technological forum for this paradigm shift in the 90s is easy to imagine. Intimate computers (ICs) will be commonplace—Dynabook-like

and even smaller. The clumsy analog cellular telephone of today will quickly be replaced by digital cellular packet-switching in which all media can be equally well represented, and this mechanism will be an obligatory component of one's IC. The most dreary projections ensure at least 100 MIPS per user and it will likely be much more, some in the form of special "beyond the curve" MIPS for special tasks such as real-time 3-D graphics and speech recognition.

The only stumbling place for this onrushing braver new world is that all of its marvels will be very difficult to communicate with, because, as always, the user interface design that could make it all simple lags far, far behind. If *communication* is the watchword, then what do we communicate with and how do we do it? We communicate with:

- Our selves
- Our tools
- Our colleagues and others
- Our agents

Until now, personal computing has concentrated mostly on the first two. Let us now extend everything we do to be part of a *grand collaboration*—with one's self, one's tools, other humans, and increasingly, with *agents*: computer processes that act as guide, as coach, and as amanuensis. The user interface design will be the critical factor in the success of this new way to work and play on the computer. One of the implications is that the "network" will not be seen at all, but rather "felt" as a shift in capacity and range from that experienced via one's own hard disk.

Another less obvious implication is that electronic mail will eventually be a very secondary medium of communication—used primarily when schedules don't mesh, for general dissemination of information, and for those situations in which independent thinking is required, particularly at the onset of projects. Network services may very well start with e-mail, but their strategy will be of much wider scope, so that several years further on, e-mail will simply be seen as a seamless part of a much wider way to amplify the user's endeavors.

Another critical strategic element is to design the services as a testbed for future technologies. This is particularly important given the lack of developed precursor technologies. As an example, it will be some years before English understanding (via typing, let alone via speech) will be at a level to sustain general conversations and understand general commands as one would give to an assistant. But much can be done now with limited English understanding in the medium of electronic mail where real-time replies are not expected and the context is restricted: registering complaints, answering requests for information about products, and so forth.

Because we design and control the user interface we can set up situations in which the system can seem much more capable than it really is without having to say: "Ignore that man behind the curtain."

Why agents at all? Well, for the same reasons that humans have used each other for agents since social interaction through communication became one of the main human traits. There are simply many areas where manipulative tools don't apply or take too long. An example is *NewsPeek*, a system I and others designed at MIT almost a decade ago. As later perfected, *NewsPeek* stays up all night looking for the newspaper you would most like to read at breakfast. It logs into a half dozen information systems including NEXUS, AP, *The New York Times*, etc., looking primarily for topics of interest to you, but also those that are especially interesting to humans in general. It reads the articles to the best of its ability—when it comes across a famous name, like "Mitterand," it finds his picture on a video disk that has 45,000 pictures of famous people—it finds maps of places mentioned on another video disk. It redefines the meaning of news—the major headline might be "Your 3:00 Meeting Is Cancelled Today" because one of its sources is your own electronic mail and it wants to bring important things it finds there to your attention. A sidebar might read "Your Children Slept Well Last Night."

Manipulative tools are simply not up to these and similar tasks—they require an agent. Another example is the situation in a large repository of information such as the Library of Congress in which there are many tools—such as the card index (the size of your home town library!) and various computer systems—but in which all transactions are carried out by human agents they supply to you when you walk in the door. The human agents are experts in not just the content of the Library, but also *the strategies of the library*. It is in these two main areas—tasks that can and should be done while you are doing something else, and tasks that require considerable strategy and expertise—that agents will gain ascendancy over tools.

The Mac already has a simple agent—it's the alarm clock desk accessory that gives a "bong" at the desired time. Simple and safe. What if we could get the alarm clock to trim the size of our hard disks periodically by deleting files? It could be done easily with HyperCard. It would still be simple to state the requirements, but we now have a very different perspective on safety. We would not like to give any current end-user programming system such a task—even if it could carry it out—without there being considerably more mechanism to explain, to make back-ups, to check and recheck. In short, we quickly want agents to have much of the common sense we would expect from a person, and to be able to learn more. It will not be an agent's manipulative skills, or even its learning abilities, that will get it accepted, but instead, its safety and ability to explain itself in critical situations.

Thus, agent development will move in two directions and later will be reconciled. Much of end-user programming in the near term will be agent-based, but only in domains in which well-meant screw-ups are not disastrous. In a serious operating system in which "delete" means simply "put this name on a list for archiving," and "undelete" is easy, these "manipulative" agents can be allowed considerable freedom. Still, there is a limit to how much remaking of a world any user wants to go through, and these agents are really not meant to be pervasive.

The second direction will move more slowly, as it has to take care of two major situations: those in which undo is hard or impossible, and those in which the user's problems are difficult to solve. The first category includes not just problems of deletions and the like, but also more personal interactions. It would not do for an agent to randomly insult clients with whom it is trying to make appointments. It is indeed hard to undo an insult!

My point here is not so much that agents are the next big direction in user interface design—that has been clear for a decade—but that this next step from the manipulated tool to the managed process will necessarily be much larger than that of the "glass teletype" to the Mac. One reason tools are easier is because they are like amplifying mirrors—they reflect and increase the user's own intelligence. An ocean vessel a mile long can be run by one tiny person because it has been designed as a continuation of the pilot's extensions into the universe. The creation of autonomous processes that can be successfully communicated with and managed is a qualitative jump from the tool—yet one that must be made. And user interface design can help the transition.

At the most basic level the thing we most want to know about an agent is not how powerful it can be, but how trustable it is. In other words, the agent must be able to explain itself well so that we have confidence that it will be working on our behalf as a goal sharer rather than as a demented genie recently escaped from *The Arabian Nights*. A really well done explanation facility will be needed regardless of whether the agent is instructed in rules, gestures, or English; whether it is anthropomorphic or tabular. This is an important and long-standing area of user interface design with disappointingly few workers.

Well, there are so many more new issues that must be explored as well. I say thank goodness for that. How do we navigate in once-again uncharted waters? I have always believed that of all the ways to approach the future, the vehicle that gets you to the most interesting places is Romance. The notion of tool has always been a romantic idea to humankind—from swords to musical instruments to personal computers, it has been easy to say: "The best way to predict the future is to invent it!" The romantic dream of "how nice it would be if . . . " often has the power to bring the vision to life. Though the notion of management of complex processes has

less cachet than that of the hero singlehandedly wielding a sword, the real romance of management is nothing less than the creation of civilization itself. What a strange and interesting frontier to investigate. As always, the strongest weapon we have to explore this new world is the one between our ears—providing it's loaded!