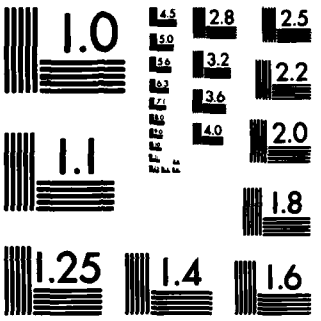
AD-A150 607 PRESENTATION AND FORM IN USER INTERFACE ARCHITECTURE (U) IBM THOMAS J WATSON RESEARCH CENTER YORKTOWN HEIGHTS NY J M CARROLL 31 AUG 83 RC-10144 1/1

UNCLASSIFIED F/G 9/2 NL



RC 10144 (#45092) 8/31/83
Computer Science/Cognition 7 pages

Research Report

PRESENTATION AND FORM IN USER INTERFACE ARCHITECTURE*

John M. Carroll
Computer Science Department
IBM Watson Research Center
Yorktown Heights, NY 10598

IBM Research Division
San Jose · Yorktown · Zurich

RC 10144 (#45092) 8/31/83
Computer Science/Cognition 7 pages -1-

PRESENTATION AND FORM IN USER INTERFACE ARCHITECTURE*

John M. Carroll
Computer Science Department
IBM Watson Research Center
Yorktown Heights, NY 10598

Abstract: It is suggested that the formal organization of function in a user interface architecture can be treated as a general-level issue, possibly amenable to codification as design guidelines, but that the presentation of function via interface elements like icons and menus must be treated on a case by case basis. Additional keywords: user interface.

This brief thought paper was invited by BYTE magazine for a special issue on user interface architecture.

Among the many trends in end-user application software are two that seem to chafe one another. On the one hand, software is getting more complex: the word processors of only a few years ago, at that time seen as a thrilling glimpse of the future, are now being superseded by office work stations with more powerful and more diverse function. The now-mundane possibilities for real-time text reformatting and interactive spreadsheets have been outdated by alternate fonts for text, voice annotation, high-quality graphics -- integrated with text, multiple concurrent tasks, and rumors of "unrestricted" voice and handwriting recognition.

Yet on the other hand, there is a gathering momentum toward increased software usability, an enormous and rapidly growing emphasis on ease of learning and ease of use directed at that "newly discovered" software component, the user interface. Unhappily, the two trends can often be at odds. If function were to be fixed, progress in the quality of user interfaces would be certain. For even without a deeply principled understanding of human-computer interaction, the problem could be attacked on a purely empirical (case by case) basis. But as functional complexity increases, the feasibility of this kind of approach recedes.

The upshot of the conflict between these two trends is not a matter of possibilities. It is a tangible part of the current reality of end user systems. People using these systems are frequently overwhelmed. They can lose track of where they are and of what they are trying to do. They can produce tangles of errors and ad hoc recovery behaviors that no manual (on-line or off) can anticipate or analyze in useful depth (see reference 5). Nor is this just an issue for novices. More experienced users, also perhaps overwhelmed by available function, tend to asymptote at a level of mediocre performance -- inured to nonoptimal use of more powerful function (see reference 10).

How are we to pursue quality in the user interface, given this situation? The answer, obviously, is "any way we can". One direction we can pursue concerns itself with the specific nature of interface elements -- the objects and actions of which the interface consists and the interface concepts, often called "metaphors", into which these primitive elements are organized. I will refer to work of this sort as being at the level of architectural *presentation*.

A second direction we can pursue specifically eschews the particulars of interface elements. It is concerned with architectural *form*, rather than presentation. The focus here is on how portions of a system's function are interrelated in typical user scenarios, but not on specifically how each piece of function is represented to the user in the interface. An example issue of architectural form is the contextual dependence of function pieces. In an interface with high contextual dependence, the range of available function is strictly fixed by the current state the user is in. The hierarchical access of function, typical of menu interfaces, is a common example of contextual dependence (for example, property sheets in the STAR-type interface; see reference 11). In an interface with low contextual dependence, any function can be accessed from any state, in the same manner and with the same effects.

A current example of an architectural presentation issue is the contrast between graphically conveyed interface metaphors, as in the use of display icons for interface actions and objects, and nongraphically conveyed interface metaphors, as in the use of metaphoric vocabulary for these actions and objects (e.g., mail, print, document, file). Other examples include the availability and nature of command-driven dialog, of prompted specifications and/or selections via menus (etc.), and of status and error feedback.

The distinction between architectural form and architectural presentation, I suggest, has important implications for the organization of research effort on user interface issues, and in particular, for the development of user interface "guidelines". My claim is that there are principles of architectural form amenable to codification as guidelines, but that there are no such principles of architectural presentation. I recognize that this claim will be controversial, and that my arguments for it will be incomplete. My purpose here is to initiate and to focus controversy, not to settle it.

Consider our examples: It seems that the issue of contextual dependence can be simply and generally resolved in a guideline: maximize contextual dependence in a user interface in order to facilitate interaction in typical user scenarios. As far as I know, following this principle optimizes for both ease of learning and ease of use. The only trade-off is that atypical user scenarios would require circumventing context dependence. The best we can ever do is the greatest good for the greatest number. But now consider the question of whether an interface metaphor should be couched as a display icon or as a labelled soft key? I don't think there is or will be a general principle to cover this case. In the balance of the paper, I examine two illustrative cases of architectural form and presentation.

Architectural Form: An Example of Staged Function.

Some of our recent research work has examined techniques for "staging" the presentation of function to new users. A staged user interface architecture makes it possible to turn off layers of function, such that the basic application scenarios can be run by the user with no frills, but so that the advanced function can easily be engaged when requested. A user interface designed this way can always be conveniently modified to be simpler or more complex by merely turning layers of function off or on.

There are two reasons why this might be a good idea: First, it deals simply and generally with the troubling impact that increasing functional complexity can have on interface quality. The remedy is to block off enough of the function to make the system seem simple. Indeed, the pitfall of prematurely and inappropriately accessing advanced function is a common new user error, and may well lay the ground work for later timidity on the part of more experienced users when it comes to exploring advanced function. Second, by having the advanced function available, this scheme provides a bridge for the user between from mastery of the core application function and mastery of the complete function. Insofar as things in fact can work this way, we have provided a simple and general solution to the conflict between increasing function and user interface quality.

In order to experiment with this idea, we designed a series of modifications in the user interface of a commerical word processing system. David Boor and I managed to define a coherent simpler level of function which included only document creation, revision, and printing function, and which specifically blocked the seven or eight most devastating new user errors which Robert Mack, Scott Robertson and I had observed in a prior study of people trying to learn the full system. Blocking the paths to these errors involved further restrictions on the creation, revision, and printing function. Essentially, we imposed an alternate architectural form on the system by brute force. The modification we obtained I will call the Training System.

Physically the Training System looked exactly like the Full System. All of the menus and other displays were exactly the same. However, when a user selected one of the error-provocative choices, options, or functions we had isolated, the Training System displayed the message "X is not available on the Training System", where X was the name of the selection. Control stayed with the current state, and the user was immediately free to make another

selection. The error consequences had been "blocked". The user could see the advanced function, and even try it -- only to be told it was not yet available, but the user would not suffer the penalty such self-initiative often carries. In the Full System, in the same situation, the user's selection would have triggered actual function, and in most cases led quickly to trouble.

Subsequently, and in collaboration with Caroline Carrithers, Jim Ford, Georgia Gibson, and Penny Smith-Kerker, I have experimented with the Training System in a series of studies reported and to-be-reported elsewhere (see for example, reference 3). Part of what we found is good news, but not altogether surprising news: novices can learn basic word processing skills several times faster if they don't have to spend time recovering from the errors of prematurely and inappropriately accessing advanced system function. In particular, the Training System users were able to reliably get to the Typing Display and begin concrete work in less than half the time as the best-performing group of learners using the Full System.

There was evidence that this advantage is more than merely a matter of reducing error time. The Training System users, at the end of our experiment, were able to type and print out a simple letter more than twice as fast as the Full System users. The Training System users reduced the proportion of their time spent on errors almost 50% more than the Full System users over the course of the experiment. And more than 90% of this improvement was due to their spending less time on the errors that were *not* blocked in the Training System itself. Hence, the advantage of the Training System is not merely a matter of blocking off errors and then observing that people spend less time on them. That is, they seemed to be more successfully developing an understanding of the system which allowed them to *avoid* all kinds of errors, and not merely to rely on the Training System to block the consequences of a subset of possible errors.

Finally, and indeed at the very end of the experiment, we administered a system concepts test and a work attitude test to our experimental subjects. The Training System people did better on both tests, indicating that they had learned more about the system, and that they felt relatively better about work in general after the experience.

What lessons can we draw from the training system work? First, we can draw some fairly specific lessons with regard to the design of training systems. The novices using our Training System were given opportunities to see where the advanced function was and to make errors, but they were protected from the direct consequences and side effects of making the errors. Nevertheless, they learned to discriminate errors from nonerrors more successfully than their Full System counterparts. Turning things around, the Full System learners were given more punishment for making errors, but their learning was impaired rather than facilitated relative to the Training group. The simple implication is that negative reinforcement has no useful role to play in the user interface. No, end user software doesn't have to hurt. If it does, it's bad software.

A second lesson we can draw from this work pertains more generally to the issue of architectural form. The system we studied was derived from a commercial system by ad hoc surgery -- long after the original design had been set in silicon. We can imagine however that when the original architecture of a machine is developed, provisions can be made for the sort of function subsetting we had to graft on by brute force. This amounts to a user interface guideline that the core of an application be itself a coherent application -- and in turn perhaps that the secondary function, along with the core, constitute a coherent application, etc. This conclusion has two nice properties: first, there is empirical evidence supporting it, and second, the recommendation it makes is simple and general.

A question that remains is how users will go from the core function all the way to the full function. We know that in the real world experienced users sometimes subset themselves and never become experts on systems they use routinely (see reference 10). In our simple case study, there were only two levels of system complexity, and in the one case in which we have examined users switching from one level to the next, users were told explicitly to switch. However, the intent in the Training System is to motivate the user to interpret increasing (but managable) functional complexity as challenge. Precisely this idea lies at the heart of many computer games (see references 2 and 9). Our hope was that seeing, and even harmlessly trying, the advanced but disabled options would help motivate the user to want to try these options again when they have become active. This remains an untested aspect of the Training System.

Architectural presentation: Problems inside of problems.

Having seen that simple and general principles can be developed at the level of architectural form, we now turn to consider architectural presentation. It seems to me that the situation is rather different. Indeed, I want to suggest that improving the quality of a user interface at the level of architectural presentation is *never just* a matter of applying simple and general principles, and unlikely to ever be accomplished by brute force. Architectural presentation is a *design* domain par excellence; as such, it is not amenable to deductive analysis (see reference 6). Rather it is fundamentally a matter of iteratively refining and developing a set of idiosyncracies.

Unfortunately, there can be no demonstrable arguments either way for such a sweeping claim. But this does not diminish the importance of the question, as even a tentative determination could help organize and allocate effort in user interface development work. I will try to illustrate the argument by discussing some recent interface metaphors for office systems.

Most simply, the motivation for developing user interface metaphors is to build upon what the user already knows, and thereby to reduce what will be conceptually novel. A large proportion of the intended users of word processors are professional typists. They know quite a lot about typewriting but not necessarily anything about computers. Analogous points can be made for potential users of electronic spreadsheets, and various other applications. Such observations suggest a simple design idea that has been very successfully exploited: help people learn and use novel systems by inviting them -- via the interface -- to engage their prior non-computer knowledge (see reference 7). The current popularity of interfaces that deliberately suggest typewriters, spreadsheets, and desk tops, is good evidence that there is something roughly right about the metaphor approach. However, I want to question whether metaphor is a simple issue with a general principle (such as "maximize contextual dependence" or "layer advanced application function"). There are two reasons for my worry.

First, metaphors are inevitable in human thought and, while they can be a source of "insight" and "savings", they can as often be a source of interference and confusion. When we focus discussion on limelighted success stories like "a word processor is a typewriter", we overlook the many classic examples of metaphor-induced troubles. Indeed, psychologists have a special term to refer to the interference of prior knowledge on problem solving activity: "functional fixedness".

An example is an experiment in which people were asked to mount candles on a vertical screen (see reference 1). They were each given a small cardboard box containing candles, thumbtacks and matches. A correct solution is to mount the candles on the boxes (by melting a little wax) and then to mount the boxes on the screen (with tacks). Boxes are typically containers, not platforms, and this interferes with the required insight. When the *materials* for

the problem were presented to the subject placed inside the boxes (reinforcing the container interpretation), only 10 percent of the participants could solve the problem. (When the boxes were presented empty along with the other materials, almost 90 percent solved the problem.)

Even the typewriter metaphor has its problematic side. Users we have studied were often quite confused by the fact that keys like Spacebar, Backspace, and Carrier Return insert blank spaces and line breaks, instead of merely moving the typing point (like on a *real* typewriter). People often balk at instructions like "backspace to erase" or "type to insert". And this can cost them much failure and frustration. They often try to change margins and tabs, when doing so is needless because of system defaults and dangerous because of the risks of getting tangled in advanced function (see reference 8). But the point is not to avoid metaphors, for this is not possible. Engaging prior knowledge in the service of present behavior and thought is a fundamental cognitive process. The point is that this process amounts to a trade-off of blocks and insight, of confusions and savings.

The second reason interface metaphors may never become a matter of simple and general principles is that they are often supremely paradoxical. They often act as conceptual aids as much because they *mismatch* their targets as because they match. Pressing character keys elicits glowing dots on a TV screen rather than of lines of ink on a paper; these are really very different effects. And typing over characters on the screen replaces the prior characters or inserts the new characters, although *both* outcomes are unpredictable on the basis of literal metaphor projection. Indeed, given a simple view of metaphor, it is remarkable that neither of these metaphor misfits has a very troubling consequence for learners. In fact, encountering these misfits can afford a concrete opportunity for developing an enhanced understanding of the electronic medium (e.g., the concept of dynamic storage). It has been argued that this paradoxical aspect of metaphor can be more important to new users than literal similarity (see reference 4).

These two properties of metaphors raise a host of questions. When is the metaphor trade-off favorable? When will metaphor mismatches be cognitively stimulating? And these questions -- which we cannot resolve in a general way, we can also not dismiss, for aspects of metaphor pervade virtually all thought, and certainly any user interface. Indeed, the very notion "user interface" implies that what the user is seeing and conceptualizing is something at least one step removed from what the system is "really" doing. Adding iconic objects and actions may make the desk top metaphor more explicit, but iconic interface entities are not necessary to suggest the metaphor in the first place. Merely describing a system as an "office application system" will have already brought to mind rich and diverse physical office metaphors. What is truly staggering, to me anyway, is how little more than this we can say on a purely analytical basis.

For having recognized that user interface metaphors are complex and unavoidable design trade-offs, we really can say little more than "try some out, test them, and try some more". The specifics can never be deduced. For example, suppose that a document removed from a folder (directory) to be edited is not returned to the folder after editing. The physical office metaphor suggests that the document should then be "left out" (e.g., on a metaphorical desktop), to be returned to the folder only when the user explicitly moves it there. But what about printing? Suppose that a document removed from a folder to be printed is not explicitly returned to the folder afterwards? Should the document remain on the metaphorical printer's metaphorical paper table? Or should the printed document be automatically refiled?

I think both choices are wrong; there is no simple solution. If the document is automatically refiled, the consistency between editing and printing is compromised. If the document is not automatically refiled, it is very likely that it will be forgotten and left in the printer. After all, when a document is sent to the printer, the users attention ultimately shifts to the printer

(not an icon or object label, but the physical thing). What happens when the next document is left in the printer? We can reason out solutions to such architectural presentation issues if we wish. However, these well-reasoned solutions will not always be right. The trade-offs and interrelations are too rich and subtle; too often the key factors are completely idiosyncratic to a particular system or application. When one problem is "solved" another appears whose very existence depends on the prior "solution".

The world-view of user interface architecture that I have been pushing here makes rather different prognoses for form and presentation. And these entail a view of where things ought to be going. Architectural form, it seems to me, is very much a matter of general principles. In this domain, the user interface guidelines every designer longs for may actually exist. For this reason, it may make sense to direct research on matters of architectural form at general-level principles. Why do less than we might? Architectural presentation, however, seems less amenable to such a treatment. Presentation issues, I think, will remain almost purely empirical case-by-case problems to be resolved by prototyping and user testing. If this world-view is correct, then the distinction between architectural form and presentation could be an important guide in planning research on user interface quality.

Note

*I am grateful to Jeff Kelley, Clayton Lewis, and Mary Beth Rosson for commenting helpfully on an earlier draft of this paper.

References

(1) Adamson, R.E. Functional fixedness as related to problem solving. *Journal of Experimental Psychology,* 1952, *44,* 288-291.

(2) Carroll, J.M. The adventure of getting to know a computer. *Computer,* November 1982, 49-58.

(3) Carroll, J.M. and Carrithers, C. Blocking user error states in a training system. Submitted to *ACM Communications.*

(4) Carroll, J.M. and Mack, R.L. Metaphor, computing systems, and active learning. IBM Research Report, RC 9636, 1982.

(5) Carroll, J.M. and Mack, R.L. Learning to use a word processor: By doing, by thinking, and by knowing. In J. Thomas and M. Schneider (Eds.) *Human factors in computer systems.* Norwood, NJ: ABLEX, 1983.

(6) Carroll, J.M. and Rosson, M.B. Behavioral specifications as a tool in iterative development. In H.R. Hartson (Ed.) *Advances in Human-Computer Interaction.* Norwood, NJ: ABLEX, forthcoming.

(7) Carroll, J.M. and Thomas, J.C. Metaphor and the cognitive representation of computing systems. *IEEE Transactions on Systems, Man, and Cybernetics,* 1982, *SMC-12,* 107-116.

(8) Mack, R.L., Lewis, C.H., and Carroll, J.M. Learning to use word processors: Problems and prospects. *ACM Transactions on Office Information Systems,* 1983, *1,* 254-271.

(9) Malone, T.W. Toward a theory of intrinsically motivating instruction *Cognitive Science,* 1981, *4,* 333-369.

(10) Rosson, M.B. Patterns of experience in text editing. CHI '83 Conference on Human Factors in Computer Systems, Proceedings. Boston, MA: December 12-15, 1983.

(11) Smith, D., Irby, C., Kimball, R., Verplank, B., and Harslem, E. Designing the STAR interface. *Byte,* April 1982, 242-282.

END

FILMED

3-85

DTIC