ABOUT GRAPHICAL USER INTERFACES

Here are some excerpts from an article[REQ6] about GUI. The article looks back over the development of Star, an early GUI system. We've added some comments in this typeface. Read the article, and the comments, critically; are all the assertions valid ? It's also instructive to notice how well, or how badly, the GUIs you know match up with the principles presented.

Desktop metaphor. Star, unlike all conventional systems and many window- and mouse-based ones, uses an analogy with real offices to make the system easy to learn. This analogy is called "the Desktop metaphor." To quote from an early article about Star:

> Every user's initial view of Star is the Desktop, which resembles the top of an office desk, together with surrounding furniture and equipment. It represents a working environment, where current projects and accessible resources reside. On the screen are displayed pictures of familiar office objects, such as documents, folders, file drawers, in-baskets, and out-baskets. These objects are displayed as small pictures, or icons.
>
> The Desktop is the principal Star technique for realizing the physical office metaphor. The icons on it are visible, concrete embodiments of the corresponding physical objects. Star users are encouraged to think of the objects on the Desktop in physical terms. You can move the icons around to arrange your Desktop as you wish. (Messy Desktops are certainly possible, just as in real life.) You can leave documents on your Desktop indefinitely, just as on a real desk, or you can file them away.

Having windows and a mouse does not make a system an embodiment of the Desktop metaphor. In a Desktop metaphor system, users deal mainly with data files, oblivious to the existence of programs. They do not "invoke a text editor," they "open a document." The system knows the type of each file and notifies the relevant application program when one is opened.

Most systems, including windowed ones, use a Tools metaphor, in which users deal mainly with applications as tools. Users start one or more application programs (such as a word processor or spreadsheet), then specify one or more data files to edit with each. Such systems do not explicitly associate applications with data files. Users bear the burden of doing that – and of remembering not to try to edit a spreadsheet file with the text editor or vice versa. User convention distinguishes different kinds of files, usually with filename extensions (such as memo.txt). Star relieves users of the need to keep track of which data file goes with which application.

SunView is an example of a window system based upon the Tools metaphor rather than the Desktop metaphor. Its users see a collection of application program windows, each used to edit certain files. Smalltalk-80, Cedar, and various Lisp environments also use the Tools metaphor rather than the Desktop metaphor.

This is not to say that the Desktop metaphor is superior to the Tools metaphor. The Desktop metaphor targets office automation and publishing. It might not suit other applications (such as software development). However, we could argue that orienting users toward their data rather than toward application programs and employing analogies with the physical world are useful techniques in any domain.

"Orienting users towards their data" fits in with our earlier comments (*PEOPLE AND COMPUTERS*) on how people view their work. Notice that the people considered in this discussion are not computists – and that neither the Macintosh nor the Windows system was directed at computists. Would some different style of interface be appropriate for, say, programmers ?

The disadvantage of assigning data files to applications is that users sometimes want to operate on a file with a program other than its "assigned" application. Such cases must be handled in Star in an ad hoc way, whereas systems like Unix allow you to run almost any file through a wide variety of programs. Star's designers feel that, for its audience, the advantages of allowing users to forget about programs outweighs this disadvantage.

Generic commands. One way to simplify a computer system is to reduce the number of commands. Star achieves simplicity without sacrificing functionality by having a small set of generic commands apply to all types of data: Move, Copy, Open, Delete, Show Properties, and Same (Copy Properties). Dedicated function keys on Star's keyboard invoke these commands. Each type of data object interprets a generic command in a way appropriate for it.

Such an approach avoids the proliferation of object-specific commands and/or command modifiers found in most systems, such as Delete Character, Delete Word, Delete Line, Delete Paragraph, and Delete File. Command modifiers are necessary in systems in which selection is only approximate. Consider the many systems in which the object of a command is specified by a combination of the cursor location and the command modifier. For example, Delete Word means "delete the word that the cursor is on."

Modifiers are unnecessary in Star because exact selection of the objects of commands is easy. In many systems, the large number of object-specific commands is made even more confusing by using single-word synonyms instead of command modifiers for similar operations on different objects. For example, depending upon whether the object of the command is a file or text, the command used might be Remove or Delete, Duplicate or Copy, and Find or Search, respectively.

Careful choice of the generic commands can further reduce the number of commands required. For example, you might think it necessary to have a generic command Print for printing various things. Having Print apply to all data objects would avoid the trap that some systems fall into of having separate commands for printing documents, spreadsheets, illustrations, directories, etc., but it is nonetheless unnecessary. In Star, users simply Copy to a printer icon whatever they want to print. Similarly, the Move command is used to invoke Send Mail by moving a document to the out-basket.

Of course, not everything can be done via generic commands. Some operations are object-specific. For example, a word might use italics, but italics are meaningless for a triangle. In Star, object-specific operations are provided via selection-dependent "soft" function keys and via menus attached to application windows.

Direct manipulation and graphical user interface. Traditional computer systems require users to remember and type a great deal just to control the system. This impedes learning and retention, especially by casual users. Star's designers favored an approach emphasizing recognition over recall, seeing and pointing over remembering and typing. This suggested using menus rather than commands. However, the designers wanted to go beyond a conventional menu-based approach. They wanted users to feel that they are manipulating data directly, rather than issuing commands to the system to do it. Star's designers also wanted to exploit the tremendous communication possibilities of the display. They wanted to move away from strictly verbal communication. Therefore, they based the system heavily upon principles that are now known as direct manipulation and graphical control.

Star users control the system by manipulating graphical elements on the screen, elements that represent the state of the system and data created

by users. The system does not distinguish between input and output. Anything displayed (output) by the system can be pointed to and acted upon by the user (input). When Star displays a directory, it (unlike MS-DOS and Unix) is not displaying a list of the names of the files in the directory, it is displaying the files themselves so that the user can manipulate them. Users of this type of system have the feeling that they are operating upon the data directly, rather than through an agent – like fetching a book from a library shelf yourself rather than asking someone to do it for you.

A related principle is that the state of the system always shows in the display. Nothing happens "behind the user's back." You needn't fiddle with the system to understand what's going on; you can understand by inspection.

One of Star's designers wrote

> When everything in a computer system is visible on the screen, the display becomes reality. Objects and actions can be understood purely in terms of their effects upon the display. This vastly simplifies understanding and reduces learning time.

We are not entirely sure about the principle suggested here. In fact, you can't show everything on the screen, and the state isn't always evident. Suppose that, for some reason, the editor programme has disappeared; now what happens when you select a file for editing ? If you know nothing about the existence of the editor, how will you understand any imaginable error message ? The danger of people believing that what they see on the screen is real is that it encourages a belief that if you can't do it on the screen then it can't be done – the desktop metaphor becomes a very limiting bound to your imagination, not a helpful way of representing something with much richer behaviour.

An example of this philosophy is the fact that, unlike many window-based computer systems (even some developed at Xerox), Star has no hidden menus – all available menus are marked with menu buttons.

Icons and iconic file management. Computer users often have difficulty managing their files. Before Star existed, a secretary at Xerox complained that she couldn't keep track of the files on her disk. An inspection of her system revealed files named memo, memo1, memo071479, letter, etc. Naming things to keep track of them is bothersome enough for programmers, but completely unnatural for most people.

Star alleviates this problem partly by representing data files with pictures of office objects called icons. Every application data file in the system has an icon representing it. Each type of file has a characteristic icon shape. If a user is looking for a spreadsheet, his or her eye can skip over mailboxes, printers, text documents, etc.

Furthermore, Star allows users to organize files spatially rather than by distinctive naming. Systems having hierarchical directories, such as Unix and MS-DOS, provide an abstract sort of "spatial" file organization, but Star's approach is concrete. Files can be kept together by putting them into a folder or simply by clumping them together on the Desktop, which models how people organize their physical worlds. Since data files are represented by icons, and files are distinguished by location and specified by selection rather than by name, users can use names like memo, memo1, letter, etc., without losing track of their files as easily as they would with most systems.

As bitmap-, window-, and mouse-based systems have become more common, the use of the term "icon" has widened to refer to any nontextual symbol on the display. In standard English, "icon" is a term for religious statues or pictures believed to contain some of the powers of the deities they represent. It would be more consistent with its normal meaning if "icon" were reserved for objects having behavioral and intrinsic properties. Most graphical symbols and labels on computer screens are therefore not icons. In Star, only representations of files on the Desktop and in folders, mailboxes, and file drawers are called icons.

Few modes. A system has modes if user actions differ in effects or availability in different situations. Tesler has argued that modes in interactive computer systems are undesirable because they restrict the functions available at any given point and force users to keep track of the system's state to know what effect their actions will have. Though modes can be helpful in guiding users through unfamiliar procedures or for handling exceptional activities, they should be used sparingly and carefully.

It's interesting to wonder why no one ever worried about modes in batch systems, where they were very common, and potentially even more damaging : if you got the mode wrong, there was no feedback, and the whole operation would have been completed in the wrong mode before you had any evidence of it. In practice, you said "Bother" (or words to that effect), changed one or two cards in your card file, submitted the job for processing again, and got on with something else. Provided that you were reasonably careful to keep copies of things that might be destroyed (which you learnt quickly), the impact was quite small, because you didn't have to repeat the work – it was all recorded in the punched cards.

Star avoids modes in several ways. One is the extensive use of generic commands (see above), which drastically reduces the number of commands needed. This, in turn, means that designers need not assign double-duty (that is, different meanings in different modes) to physical controls.

A second way is by allowing applications to operate simultaneously. When using one program (such as a document editor), users are not in a mode that prevents them from using the capabilities of other programs (such as the desktop manager).

A third way Star avoids modes is by using a noun-verb command syntax. Users select an operand (such as a file, a word, or a table), then invoke a command. In conventional systems, arguments follow commands, either on a command line or in response to prompts. Whether a system uses noun-verb or verb-noun syntax has a lot to do with how moded it is. In a noun-verb system such as Star, selecting an object prior to choosing a command does not put the system into a mode. Users can decide not to invoke the command without having to "escape out" of anything or can select a different object to operate on.

We commented on this point in the *PEOPLE TALKING TO COMPUTERS* chapter. We don't see why selecting a programme should put the system into a mode either. You can still decide not to do it, and select something else.

Though Star avoids modes, it is not completely modeless. For example, the Move and Copy commands require two arguments: the object to be moved and the final destination. Though less moded ways to design Move and Copy exist, these functions currently require the user to select the object, press the Move or Copy key, then indicate the destination using the mouse. While Star waits for the user to point to a destination, it is in Move or Copy mode, precluding other uses of the mouse. These modes are relatively harmless, however, because (1) the shape of the cursor clearly indicates the state of the system and (2) the user enters and exits them in the course of carrying out a single mental plan, making it unlikely that the system will be in the "wrong" mode when the user begins his or her next action.

Objects have properties. Properties allow objects of the same type to vary in appearance, layout, and behavior. For example, files have a Name property, characters have a Font family property, and paragraphs have a Justified property. Properties may have different types of values: the Name property of a file is a text string; the Size property of a character might be a number or a choice from a menu; the Justified property of a paragraph is either "on" or "off." In Star, properties are displayed and changed in graphical forms called property sheets.

Property-based systems are rare. Most computer systems, even today, allow users to set parameters for the duration of an interactive session or for the duration of a command, but not for particular data objects. For example, headings in Wordstar documents do not "remember" whether they are centered or not; whether a line is centered is determined by how the program was set when the line was typed. Similarly, directories in Unix do not "remember" whether files are to be listed in alphabetical or temporal order; users must respecify which display order they want every time they invoke the ls command .

"Property-based systems" have not always been quite as rare as is suggested. We'll remark later that older systems often carried more information about files than did others of more recent origin. It’s interesting that one of the reasons for reducing the amount of information was to make the systems easier to use !

Progressive disclosure. It has been said that "computers promise the fountains of utopia, but only deliver a flood of information." Indeed, many computer systems overwhelm their users with choices, commands to remember, and poorly organized output, much of it irrelevant to what the user is trying to do. They make no presumptions about what the user wants. Thus, they are designed as if all possible user actions were equally likely and as if all information generated by the system were of equal interest to the user. Some systems diminish the problem somewhat by providing default settings of parameters to simplify tasks expected to be common.

Star goes further towards alleviating this problem by applying a principle called "progressive disclosure." Progressive disclosure dictates that detail be hidden from users until they ask or need to see it. Thus, Star not only provides default settings, it hides settings that users are unlikely to change until users indicate that they want to change them. Implicit in this design are assumptions about which properties will be less frequently altered.

One place progressive disclosure is used is in property sheets. Some objects have a large number of properties, many of which are relevant only when other properties have certain values (see Figure 2). For example, on the page layout property sheet, there is no reason to display all of the properties for specifying running header content and position unless the user actually specifies that the document will have running headers.

Another example of progressive disclosure is the fact that property displays in Star are temporary, displayed on demand. In some systems, the properties of the current selection are displayed at all times, through codes embedded in the text or in an area of the screen reserved

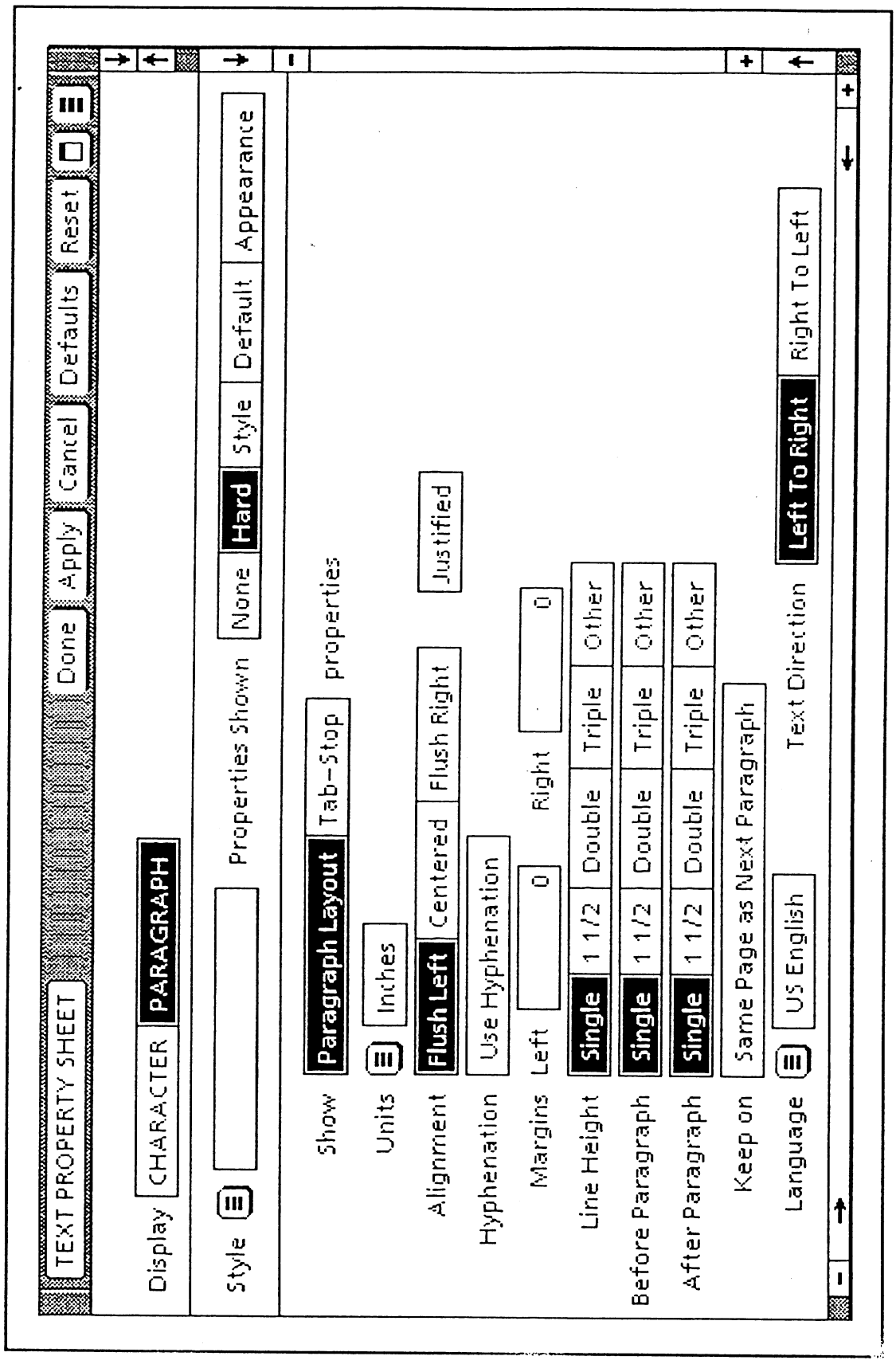

Figure 2. Progressive disclosure. Star's property sheets, like the rest of the interface, use a principle known as progressive disclosure to avoid overwhelming users with information. Usually, users don't need to see an object's properties: they only need to see and perhaps change its assigned style. Users see an object's properties only upon request. Also, even when a user sets a property sheet to show an object's properties, as shown here, some information remains hidden until the user asks to see it. For example, there is no need to clutter the property sheet here with boxes for entering numbers for "Other" values of Line Height, Spacing Before Paragraph, or Spacing After Paragraph until the user actually sets the property to "Other."

for that purpose, even though the user usually doesn't care.

A highly refined manifestation of progressive disclosure recently added to ViewPoint is *styles*, which allows users to regard document content (such as a paragraph) as having a single style rule instead of a large number of properties. Thus, styles hide needless detail from users.

Consistency. Because Star and all of its applications were designed and developed in-house, its designers had more control over its user interface than is usually the case with computer systems. Because the designers paid close attention to detail, they achieved a very high degree of consistency. The left mouse button always selects; the right always extends the selection. Mouse-sensitive areas always give feedback when the left button goes down, but never take effect until the button comes up.

Emphasis on good graphic and screen design. Windows, icons, and property sheets are useless if users can't easily distinguish them from the background or each other, can't easily see which labels correspond to which objects, or can't cope with the visual clutter. To assure that Star presents information in a maximally perceivable and useful fashion, Xerox hired graphic designers to determine the appearance and placement of screen objects.

These designers applied various written and unwritten principles to the design of the window headers and borders, the Desktop background, the command buttons, the pop-up menus, the property sheets, and the Desktop icons. The most important principles are

- The illusion of manipulable objects. One goal, fundamental to the notion of direct manipulation, is to create the illusion of manipulable objects. It should be clear that objects can be selected and how to select them. It should be obvious when they are selected and that the next action will apply to them. Whereas the usual task of graphic designers is to present information for passive viewing, Star's designers had to figure out how to present information for manipulation as well. This shows most clearly in the Desktop icons, with their clear figure/ground relationship: the icons stand by themselves, with self-contained labels. Windows reveal in their borders the "handles" for scrolling, paging, window-specific commands, and pop-up menus .

- Visual order and user focus. One of the most obvious contributions of good graphic design is appropriate visual order and focus on the screen. For example, intensity and contrast, when appropriately applied, draw the user's attention to the most important features of the display.

  In some windowing systems, window interiors have the same (dark) color as the Desktop background. Window content should have high intensity relative to the Desktop, to draw attention to what is important on the screen. In Star, window content background is white, both for high contrast and to simulate paper.

  Star keeps the amount of black on the screen to a minimum to make the selection stand out (see Figure 3).

  In most windowing systems, window headers and other areas of the screen are black, making the selection hard to find. This principle is so important that Star's designers made sure that the display hardware could fill the nonaddressable border of the screen with Desktop grey rather than leaving it black as in most systems. Star also uses icon images that turn from mostly white to mostly black when selected (see Figure 4) and allows at most one selection on the screen at a time.

- Revealed structure. Often, the more powerful the program used, the greater the distance between intention and effect. If only effect is displayed and not intention, the user's task of learning the connection is much more difficult. A good graphical interface can make apparent to the user these connections between intention and effect, that is, "revealed structure." For example, there are many ways to determine the position and length of a line of text on a page. It can be done

with page margins, paragraph indentations, centering, tabs, blank lines, or spaces. The WYSIWYG, or "what you see is what you get," view of all these would be identical. That would be enough if all that mattered to the user was the final form on paper. But what will happen if characters are inserted? If the line is moved to another page, where will it land? WYSIWYG views are sometimes not enough.

Special views are one method of revealing structure. In Star, documents can show "Structure" and/or "Non-Printing Characters" if desired (see Figure 5). Another convenient means for revealing structure is to make it show up during selection. For example, when a rectangle is selected in a graphics frame, eight control points highlight it, any of which can attach to the cursor during Move or Copy and can land or. grid points for precise

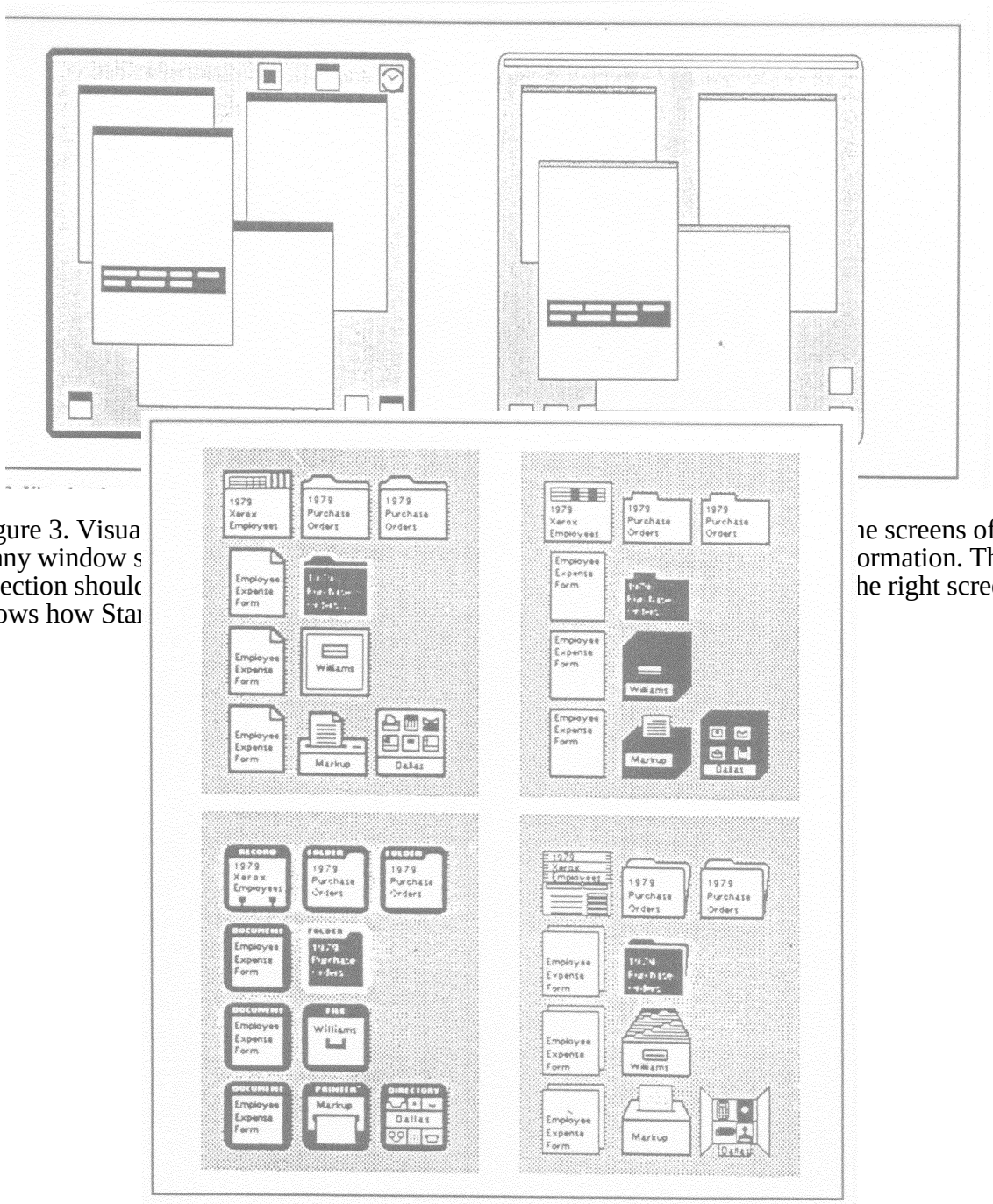

Figure 3. Visua ıe screens of many window s ormation. The selection should he right screen shows how Star

Figure 4. Visual order and user focus. Four candidate sets of icons were designed and tested for Star. A representative sample from each set is shown here. In Star, the icon selected by the user is indicated by inverting its image. Candidate icon sets in which the images are mostly white allow icons to stand out when selected. The set that best satisfies this criterion, the one on the upper left, was chosen.

alignment. The control point highlighting allows a user to distinguish a rectangle from four straight lines; both might produce the same printed effect but would respond differently to editing.

- Consistent and appropriate graphic vocabulary. Property sheets (see Figure 2) present a form-like display for the user to specify detailed property settings and arguments to commands. They were designed with a consistent graphic vocabulary. All of the user's targets are in boxes; unchangeable information such as a property name is not. Mutually exclusive values within choice parameters appear with boxes adjacent. Independent "on/off" or state parameters appear with boxes separated. The current settings are shown inverted. Some of the menus display graphic symbols rather than text. Finally, there are text parameters consisting of a box into which text or numbers can be typed, copied, or moved, and within which text editing functions are available.

Figure 2) present a form-like display for the user to specify detailed property settings and arguments to commands. They were designed with a consistent graphic vocabulary. All of the user's targets are in boxes; unchangeable information such as a property name is not. Mutually exclusive values within choice parameters appear with boxes adjacent. Independent "on/off" or state parameters appear with boxes separated. The current settings are shown inverted. Some of the menus display graphic symbols rather than text. Finally, there are text parameters consisting of a box into which text or numbers can be typed, copied, or moved, and within which text editing functions are available.

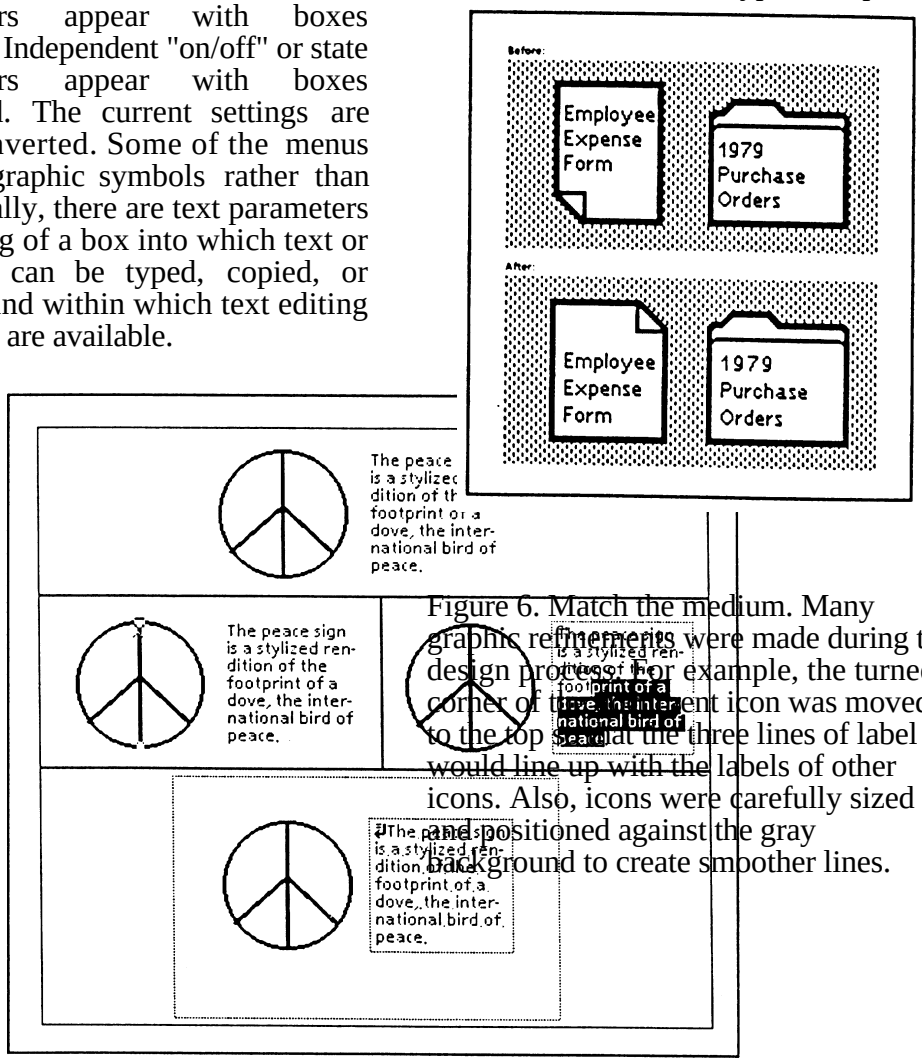

Figure 6. Match the medium. Many graphic refinements were made during the design process. For example, the turned corner of the document icon was moved to the top so that the three lines of label would line up with the labels of other icons. Also, icons were carefully sized and positioned against the gray background to create smoother lines.

Figure 5. Revealed structure. At the top is the WYSIWYG view of mixed text and graphics. The middle two panels show that structure is revealed when an object is selected. When a line segment is selected, its control points are shown. When text is selected, the text string is revealed. The bottom panel shows the effect of the Show Structure and Show Non-Printing Characters commands, which is to reveal the location of embedded graphics and text frames (dotted lines) and "new paragraph" and Space characters.

- Consistent and appropriate graphic vocabulary. Property sheets (see Figure 2) present a form-like display for the user to specify detailed property settings and arguments to commands. They were designed with a consistent graphic vocabulary. All of the user's targets are in boxes; unchangeable information such as a property name is not. Mutually exclusive values within choice parameters appear with boxes adjacent. Independent "on/off" or state parameters appear with boxes separated. The current settings are shown inverted. Some of the menus display graphic symbols rather than text. Finally, there are text parameters consisting of a box into which text or numbers can be typed, copied, or moved, and within which text editing functions are available.

- Match the medium. It is in this last principle that the sensitivities of a good graphic designer are most apparent. The goal is to create a consistent quality in the graphics that is appropriate to the product and makes the most of the given medium. Star has a large black and white display. The solutions the graphics designers devised might have been very different had the display had grey-scale or color pixels.

  A common problem with raster displays is "jaggies": diagonal lines appearing as staircases. With careful design, jaggies can be avoided, for example, by using only vertical horizontal, and 45-degree angles. Also important is controlling how the edges of the figures interact with the ground. Figure 6 shows how edges are carefully matched to the background texture so that they have a consistent quality appearance.

REFERENCE.

REQ6 : J. Johnson, T.L. Roberts, W. Verplank, D.C. Smith, C.H. Irby, M. Beard, K. Mackey : "The Xerox Star : a retrospective", *IEEE Computer* **22#9**, 11 (September 1989).

QUESTIONS.

From an examination answer : "After a week or two on the Macintoshes I felt like an expert". Comment.

From an examination answer : "... the style is consistent ... always delete by dragging to a trash can". From a word processor ? And what about ejecting a disc by dragging it to trash ?
