The principal designer of the Sapphire window manager talks about its icons and user commands in this tutorial on the screen allocation package.

The User Interface for Sapphire

Brad A. Myers
University of Toronto

Sapphire (the Screen Allocation Package Providing Helpful Icons and Rectangular Environments) is a very powerful window manager running on the PERQ personal workstation.[1] The PERQ has a high-resolution screen and special hardware to display graphics quickly. Sapphire was designed to support program development in the Accent[2] multiprocessing operating system. Companies (OEMs) that purchase PERQs may also use Sapphire in their final products, so unlike such other window managers as Star,[3] Sapphire must be general purpose and highly flexible. Sapphire is now used by quite a large community at PERQ Systems Corporation, Carnegie-Mellon University, and elsewhere. It supports a full implementation of the covered window paradigm (where the rectangular windows can overlap like pieces of paper on a desk), which is described in another article.[4] In Sapphire, windows can cover each other and can extend off the screen in any direction (and may be entirely offscreen).

All window managers can be logically divided into three functions: (1) the implementation of the low-level graphics primitives (which includes preventing graphics from showing through covered portions), (2) the presentation of pictures to the user by the window manager (such as window title lines and icons), and (3) the operations that allow the user to manipulate windows. The latter two functions of a window manager are collected under the heading *User Interface*. This article presents the user interface of Sapphire and explains why it is novel and appealing. First, however, some background material is included for those who are not familiar with window managers.

Background

Personal workstations. Interest in window management systems has expanded with the proliferation of personal computers. *Personal workstations* are personal computers that are used by only one person at a time but provide far more power than typical home computers. A personal workstation will usually have a processor that can execute over one million instructions per second, a hard disk that can hold over 20 million bytes of data, a memory of at least one million bytes, some number of input/output options, such as RS-232, Ethernet, IEEE 488, floppies, etc., and a high-performance screen that is capable of graphics. Most personal workstations currently use high-resolution bit-map screens (about 800 × 1000) where each point on the screen (called a *pixel*) is associated with one bit of memory. Each pixel can be either on or off (white or black). Many personal workstations have some sort of hardware that allows screen operations to run swiftly, and some offer color screens as an option.

Most personal workstations run an operating system, such as Unix,[5] that allows the user to run a number of different jobs (sometimes called *processes*) at the same time. For example, the user might specify that a compilation should continue to run (in the background) while the user enters the editor to work on a different file. Even with the high performance of a personal workstation, there will unfortunately always be jobs that cannot be processed instantly (i.e., fast enough so the user does not notice the delay). With multiprocessing the user does not have to wait idly for jobs to be finished, so time can be used more effectively.

0272-1716/84/1200-0013$01.00 1984 IEEE

Window managers. If a user runs more than one process in most conventional systems, the input and output from the various processes can be confusingly intermixed on the screen. For example, the output from one process may appear while another process is listing a file and be missed entirely, or the output might appear while running a screen editor or a graphics program and thereby mess up the screen. When two processes request input at the same time, the user may give the input to the wrong program. A window manager solves these problems by simulating several terminals on the same physical screen. The input and output for the different "virtual terminals" is kept separate by the window manager. Each of the virtual terminals is called a *window*. This is a simplification, however, since windows are also used in many other ways. There are several kinds of window managers, differentiated by where they allow windows to be and how the user controls the windows. All have as their basic goal allowing the user to control a number of separate activities more easily.

Since there is only one keyboard, but multiple windows, it is clear that there must be some way to identify where typing is directed. Some window managers call the window accepting input the *active* window, but this is often confusing, since many windows may be actively displaying output at the same time in such window managers as Sapphire. Therefore, in our system we coined the term *Listener* to refer to the window accepting typing, since it is "listening" to the keyboard. The user is provided with mechanisms for changing the Listener so it is possible to *talk* to any process.

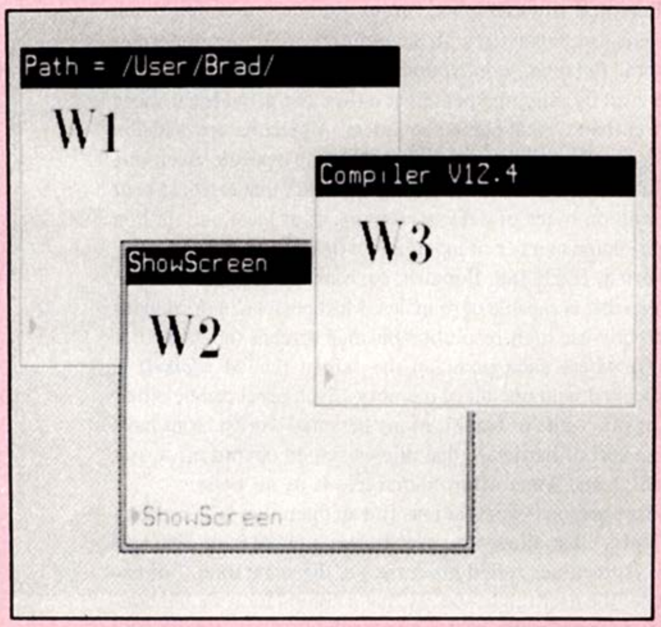

Figure 1. Three Sapphire windows. Window W3 is covering window W2, and both windows W3 and W2 are covering W1. W1 is on the "bottom" and W3 is on the "top." W2 has a gray border to show that it is the Listener (accepting user input).

The covered window paradigm. Almost all window managers require that windows be rectangular. A very simple window manager might divide the screen horizontally and/or vertically into some number of sections of fixed size. The next level of complexity is to have a number of variable-size windows that are not allowed to overlap. For example, the Cedar environment[6] from Xerox PARC CSL allows windows to be any height that fits on the screen in two vertical columns. This technique has several advantages: It can readily create windows and control their placement automatically, and it can implement graphic operations more easily. A problem in a nonoverlapping system is that it may be difficult for the user to get a window big enough to present sufficient contextual information to accomplish the task easily. Imagine trying to edit a file when you are only allowed to see three or four lines at a time! Even if it is possible to change a window's size and location, the user may feel severely restricted, since windows cannot be overlapped.

Several window managers allow the windows to overlap on the screen. The windows can be thought of as pieces of paper, and the screen can be thought of as a desk.[7] Thus a window may be on top of another window, just as one piece of paper may be on top of another piece of paper. The window that is behind is *covered* by the window on top, and the parts that are underneath the top window do not show through (see Figure 1). There is a total ordering imposed on all the windows where windows *higher* in the order cover windows that are *lower*. Windows that do not interact are still ordered. The top window is not covered by any windows, and the window that is most covered is said to be on the *bottom*. Windows may also extend off the screen in any direction, with the parts not on the screen simply not showing. The covered window paradigm was developed at Xerox PARC for DLisp[7] and Smalltalk.[8]

The advantage of this technique is that there can be a number of large windows that would not all fit on the screen if they were required to be side by side. The user can then rearrange these windows as needed to make the window of interest visible. The disadvantages are that the covered window scheme is fairly difficult to implement, and the screen may become cluttered if there are lots of windows. Clearly, a window manager implementing a covered window paradigm can simulate one without covered windows, if the user prefers.

Manipulation. For a covered window scheme to be used easily, the user must be able to manipulate the windows readily. At a minimum, the user should be able to bring a specified window, which may previously have been covered by other windows, to the top. The user must also have some way of specifying which window should be the Listener. Other operations that are useful are the ability to move a window around the screen, to change its size, and to send it to the bottom. The last is often used to locate a window that is totally covered by other windows. The typical way to find the window in many window systems is to send other windows to the bottom until the desired window appears on top.

Some window managers require that the Listener window always be on top. From an implementation point of view, this is not necessary. If the window manager supports output to windows that are covered, allowing the Listener to be covered presents no additional difficulty. The user might have the Listener window full screen, for example, yet want to copy some information from some other, smaller, window. The smaller window could then be brought to the top for inspection, while typing continued to the Listener, which would now be partially covered.

Display. One of the difficult things to implement in a covered window system is the display of text and graphics in windows that are partially covered. The covered window system must ensure that the text and graphics for one window are not displayed in the parts that are covered by other windows. Some systems handle this problem by requiring that the window getting output never be covered. Other systems, such as the Lisp Machine,[9] handle this by sending all the output to special offscreen buffers and then copying the visible portions onto the screen. This requires that the display operations be done twice, once to the offscreen buffer and then once to the screen (for portions that are visible). To avoid this overhead, Sapphire and the Blit terminal[10] divide the screen window into rectangles that may be either covered or visible. For the visible rectangles the output is directed to the screen, but for the covered rectangles the output is (optionally) directed to an offscreen buffer. This offscreen buffer is then used to regenerate the picture if the window becomes uncovered.

Pointing device. Most personal workstations come with some form of pointing device, which returns a two-dimensional value that allows the user to identify locations on the screen by simply pointing to them. The pointing device may also be used for specifying window size and position, for identifying characters in an editor, for drawing lines in a graphics program, and for transferring a picture (such as a map) into the computer by specifying points (this is called *digitizing*). Examples of pointing devices (see Figure 2) are lightpens, electromagnetic tablets, touch-sensitive surfaces, and mechanical or optical "mice."[11] Light pens are used by pointing directly to the screen, but the user moves the other devices on the desk or on a special surface, and a small picture (the *tracking symbol*), follows the movement on the screen. Some touch-sensitive surfaces are actually mounted directly on the screen, but these usually have tracking symbols too. In many personal workstations the picture for the cursor can be changed, but a common picture is an arrow pointing to the upper left (Number 1 in Figure 7).

Many pointing devices can be operated in two modes: absolute and relative. In absolute mode the position of the pointing device on a special surface specifies where the tracking symbol will be on the screen. For example, if the pointing device is in the upper left corner of the surface, then the tracking symbol will be in the upper left corner of the screen. In relative mode the actual position has no meaning; only *movements* are important. Thus, no matter where the tracking symbol is, putting the pointing device on the surface and moving it in some direction will move the tracking symbol by a proportional amount in the same direction. Relative mode is preferred by many users, since only a portion of the surface needs to be used (or accessible).

Such devices as mice can only provide relative mode. Absolute mode, however, is necessary for digitizing and other applications. A device that provides absolute mode can simulate relative mode fairly easily in software.

Pointing devices often have one or more buttons. (There are typically one to three buttons on a mouse.) Electromagnetic tables come with a pen (called a *stylus*) with one button (pressing down on the pen tip is interpreted as a button push) or a three-to-25-button puck (see Figure 2). The buttons on these devices are usually pressed by the user to tell the computer that the current position of the pointing device should be interpreted in some way. A common way to identify a window, for example, is to move the pointing device until the tracking symbol is over the desired window and then press one of the buttons.

Icons. Even with the operations specified above, it is sometimes difficult to manage a large number of windows. The screen gets cluttered, and finding a desired window is very difficult. Icons were first used as a way of representing a window by Xerox in the Star.[3] The Apple Lisa[12] and Macintosh use icons in essentially the same manner. The window seems to have been "shrunk" down until only its icon is visible. (In Sapphire the icon and the window exist at the same time, as will be explained later. This differs from the operation of icons in the Star and Lisa.) The icons at Xerox and

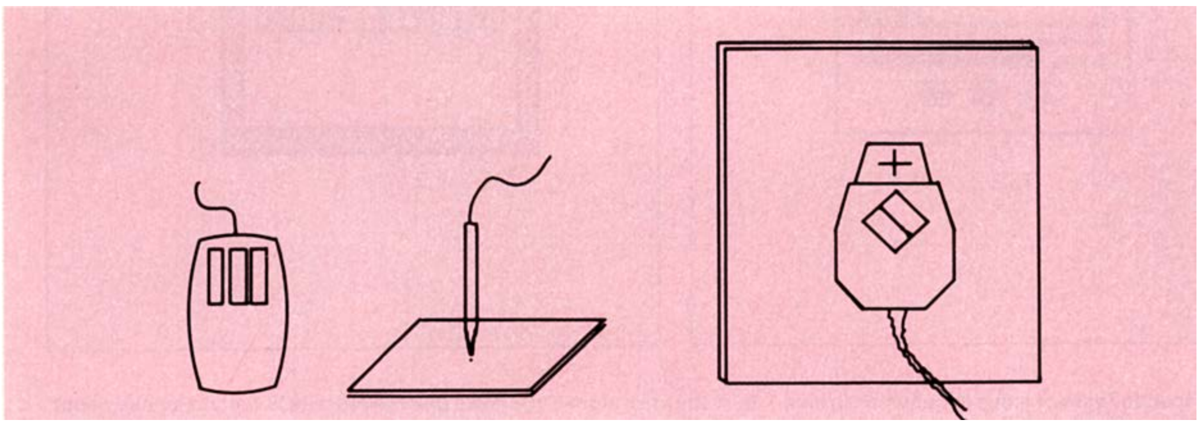

Figure 2. Examples of different pointing devices. These are not drawn to scale. The first is a typical three-button mechanical or optical mouse. The second is a stylus on an electromagnetic tablet, and the third is a four-button puck, which operates on the same type of tablet.

Apple are typically little pictures, about the size of a postage stamp, that attempt to represent pictorially the process running in the window (see Figure 3). For example, a mail program might be represented by a picture of a mailbox. When finished with the mail program, the user gives a command that shrinks the mail program's window down to its icon. The window is no longer on the screen, and the little icon is neatly put out of the way at the edge of the screen. When the mail program is needed again, the user points to the icon for the mail program (using the pointing device) and gives some command. Icons can also represent such static objects as documents, and moving one of these to a process icon causes the process to operate on the object. Icons help users understand how to operate the system because they are a metaphor of the real world. Saving a file in a directory is presented as putting a ''document'' icon in a ''folder'' icon in a ''file cabinet'' icon. As explained below, Sapphire views icons differently and has greatly expanded their functionality, so they can be used for multiple process monitoring, as well as for window control.

Presentation of Sapphire windows

Windows in Sapphire usually have title lines and borders (see Figure 1). Application programs may create windows without either, but the borders are useful for showing where the windows are, and the title lines are useful for displaying status information. For example, in a multiple directory system such as Unix the title line might contain the current directory. A window running the compiler might have the version number of the compiler and the name of the file being processed displayed in the title line. The title line is shown in reverse video at the top of the window (Figure 1).

It is important that the user be able to easily determine which window is the Listener. Other systems simply identify the Listener by which window contains the tracking symbol. Since Sapphire does not change the Listener when the pointing device is moved, some graphic method is needed. Sapphire shows the Listener by changing the border from a single hairline to two hairlines surrounding a gray area (see W2 in Figure 1). Many other systems show the Listener by

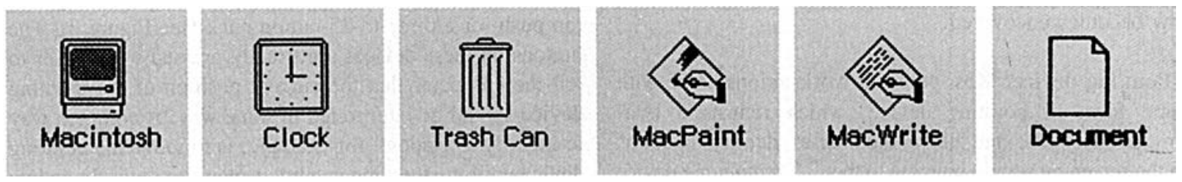

Figure 3. Samples of icons used in the Apple Macintosh computer.

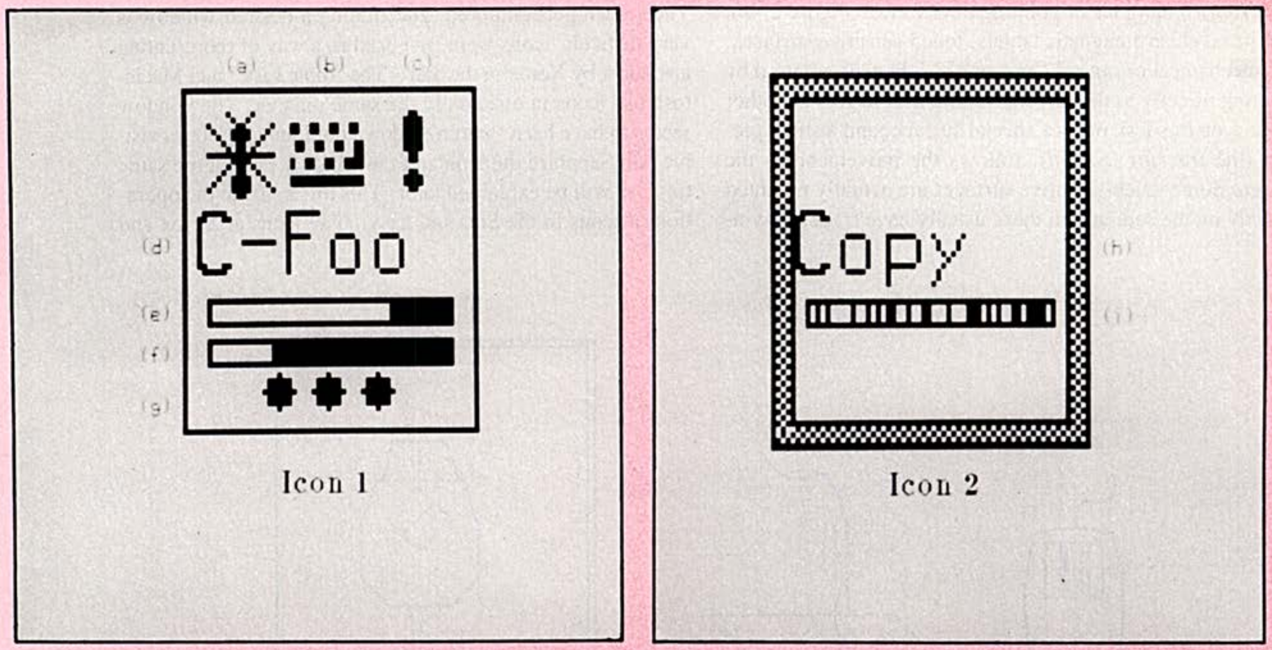

Figure 4. Expanded view of icons in Sapphire. In icon 1, (a) is the error signal, (b) shows that the process is waiting for user input, and (c) is the application-defined attention signal. (e) and (f) are two "percent-done progress bars," and (g) shows that the associated window is offscreen. Icon 2 corresponds to the Listener (the border is shown in gray), and it has its first progress bar (i) showing random progress. The process associated with icon 2 is "Copy" (h). In icon 1, the compiler has shown (d) the name of the program being processed ("C" for compiler, and "Foo" is the name of the program).

simply highlighting the title line in some manner, but this has the disadvantage that if the entire top of the Listener's window is covered or offscreen, there is no visible indication. Also, highlighting the entire border seems to make the Listener easier to find when scanning the screen. Another advantage of highlighting the border rather than the title line is that almost all windows will have borders, but an application may create a window without a title line if no status information is needed (and Sapphire does allow this). In this case, you still want to know if the window is the Listener or not.

Icons

The icons in Sapphire are very different from the icons in Lisa or the Star. Icons were introduced on the Star primarily to make the operations on the computer seem more like the corresponding operations in the office environment, and thereby make the system easier for the naive user to learn. Consequently, the icons are pictures of the operations they represent. The Lisa uses icons in the same way. This has a number of severe limitations. First, many operations do not have obvious pictorial representations. Some pictures may be ambiguous, hard to understand, or even offensive to some users. In addition, while such representations as these icons may help the novice learn the system, they may actually hinder the expert user.[13] Icons in Sapphire do not attempt to enforce a unified analogical view of the system, although application programs that use Sapphire may use icons in this manner.

The icons in Sapphire were designed with an entirely different goal in mind. Sapphire's icons are intended to enhance the user's productivity when executing multiple tasks concurrently. As was explained above, users will often multitask to increase their efficiency. However, people easily lose track of what they are doing and need aids to help plan, monitor, and control the various tasks operating at the same time. Therefore, the icons in Sapphire present six pieces of information about the process being run, as well as two pieces of information about the status of the window (see Figure 4).

Process name. First, there is the process name (this is (d) and (h) in Figure 4). The application program may optionally replace this with some other useful names. Thus, the icon for a compiler might use "C-Foo" when compiling a file titled Foo, rather than just "Compiler."

Progress bars. In the icon there are also two "percent done progress bars" (these are (e), (f), and (i) in Figure 4). Progress bars function like the giant thermometers used to mark progress in charity drives. They give the user enough information at a quick glance to estimate how much of the task has been completed. These are kept up to date by the application program (or by the system for certain operations), so the user can always check how far along the process is. For example, a compiler can report how far it is through the file. The first progress bar ((e) and (i) in Figure 4) is used by the application program and is repeated in the title line of the window underneath the text (Figure 5). The second progress bar (see (f) in Figure 4), which appears only in the icon, is used for such aggregates as command files, to tell how much of the entire job has been completed. A formal study has shown that users prefer systems with progress bars.[14]

Most applications can calculate or estimate what percentage of the work has been completed, but for those that cannot, Sapphire provides "random progress," which shows

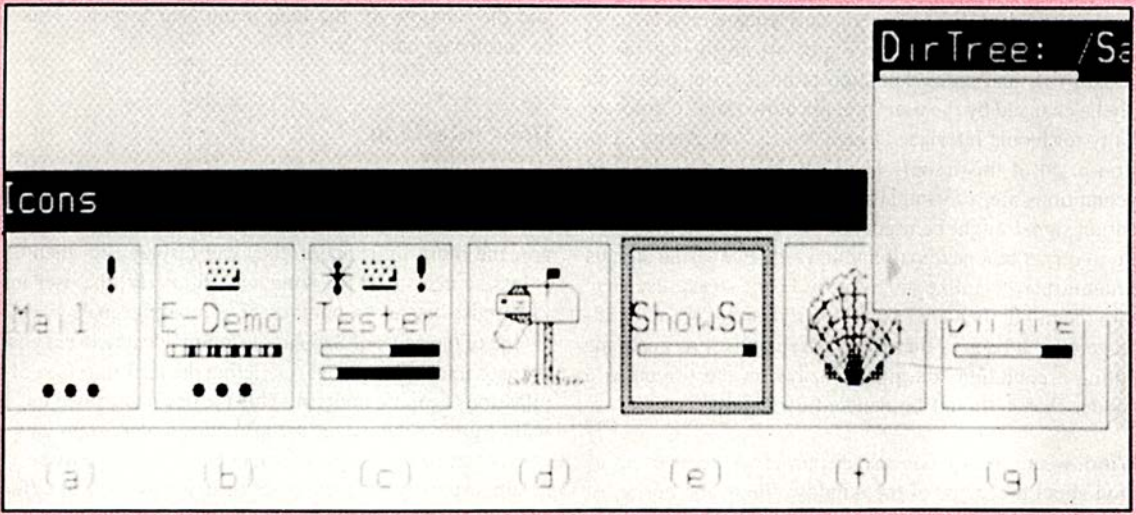

Figure 5. A typical Sapphire icon window. Note that this window can be covered by other windows. Icon (a) is for the Mail program which is using the attention signal (exclamation point) to show that new mail has arrived. The three dots show that the window is offscreen. Icon (b) is for the editor working on the file "Demo." It is waiting for user-input (the keyboard), and is showing "random progress." It is also offscreen. Icon (c) has the error signal (bug) displayed, as well as user input and attention signals. It also has two "percent-done progress bars," the first showing that Tester is about 60% complete, and the second showing that the entire job is about 20% complete. Icons (d) and (f) have application-defined pictures in them. Icon (e) has a gray border to show that it corresponds to the Listener window. Icon (g) and its window show that the first progress bar is repeated in the window title line.

that work is progressing by continually XORing vertical lines at random places along the progress bar. This displays a continually changing pattern that shows that the program is processing, but is not confused with normal progress. In Figure 4, (i) shows random progress.

Progress bars have proven to be an extremely helpful user interface feature for a number of reasons: They make the users feel better about the system because it is obvious that the program is making forward progress on requests and has not gotten into an infinite loop. Many systems present a "busy" picture, such as an hourglass, clock, or a Buddha (for patience), to show that they are computing, but since this is static, it does not show that the program has not crashed or how swiftly it is progressing toward completion. Independent of any other advantages, this lowering of the user's anxiety is an important benefit. Users can also use the progress information to estimate when jobs will be completed so they can schedule their time more effectively, either in other computer tasks or in off-computer activities. Thus, users may be more productive when provided with progress information.

A major complaint about progress bars is that it is too difficult for applications programs to compute what percent of the job they have completed. In the PERQ POS operating system,[15] however, experience has shown that most applications can provide this information with only a little work. Since the display is very low resolution, an approximation is all that is needed. Estimates of progress can usually be calculated based on the amount of an input file that has been read, for example, and (possibly) heuristics based on past performance. In some cases operating systems can provide useful information that will allow progress to be computed and displayed.[15]

The last three pieces of process information provided by Sapphire are shown as pictures in the icon. One picture tells whether there has been an error in the last activity, another tells when the process is waiting for user input, and the third is reserved for specific application-defined attention signals. The actual pictures used (see Figure 4) are a bug (a), a keyboard (b), and an exclamation point (c), but these can easily be changed by the user or application (see "Corporate Identity for Iconic Interface Design . . . " by Aaron Marcus on p. 24 of this issue). The pictures are present when the conditions are true and absent when they are false. The attention signal might be used, for example, by a mail program to report that new mail has arrived. The visual signals are nonintrusive, unlike an auditory beep, so the user can ignore them, but they are easy to see if desired. Another advantage over a beep is that if the user chooses to postpone handling a condition, the picture stays on the screen as a reminder that it should be handled eventually.

Window status. An icon also displays two pieces of information about the status of the window. First, the border of the icon for the Listener is highlighted in the same manner as the window itself (in Figure 4, icon 2 corresponds to the Listener). Second, the icon displays a small picture if the window has been moved entirely off the screen (see (g) in Figure 4).

Although the icons are small (64 × 64 pixels each), they are easy to interpret, and the information is displayed in a convenient manner. The user can simply scan the icons to deduce the state of the entire computer and easily see which processes require attention. Of course, if the default information described above is not appropriate for some processes, arbitrary pictures can be displayed in the icon (as was done by two processes in Figure 5). In this way, the icons of the Star or Lisa can be simulated by Sapphire.

Icon window. In almost all other window managers with icons an icon appears when a window is removed from the screen. Thus the window either is displayed or has been shrunk down so that only the icon shows. In Sapphire, however, it seems clear that we want icons for all windows displayed at all times, since they provide so much useful information. There is no logical difference between a window that is totally covered by other windows and one that is offscreen; both are invisible to the user. Therefore, icons in Sapphire are visible for all windows. A window may still be removed from the screen in Sapphire by simply moving it so that it is totally offscreen (see next section). The icons are therefore *associated* with a window rather than an alternative representation for it.

Since the icons and the windows they represent can be on the screen at the same time in Sapphire, it might be difficult to specify whether such operations as "move" should operate on an icon or on its window. To alleviate this problem, Sapphire groups all icons together in one window. Thus, the icons as a group can be moved around or removed altogether if the user does not like icons. The icon window can be covered by other windows, and its size and position can be changed in the same manner as any other window.

The icons in the icon window are not rearranged except on user command. Thus, when a window is deleted, its icon is deleted, but the hole is not filled until another window is created. Users will probably remember which window goes with which icon by position, so it is important not to rearrange the icons without the user's permission. Of course, there are also commands to identify the icon for a window and the window for the icon if the user forgets. This will be discussed later.

User interface

There are a number of trade-offs to be considered in designing a window manager. For example, the larger the icons are, the more information they can display, but then there is less screen space for windows. Similarly, the user interface must balance a number of competing goals.

One of the goals of Sapphire is to provide a rich and powerful user interface without restricting the user interface of applications running under it. This is important, since the user will be giving commands to the application program far more often than to the window manager. Sapphire will be used to support many different types of applications with different requirements for input and output. Therefore, Sapphire, unlike most other window managers, does not reserve any of the buttons on the pointing device exclusively for use by Sapphire. Any button pressed or released inside the Listener window is sent to the application running in that window. To give window-manager commands for a window, the user must press a button in the title line or in the icon for that window.

Input devices. Sapphire is also designed to run with a number of different input devices, one of which is a one-button stylus. This provides two constraints. First, all operations have to be possible with one button, and second, the point position cannot be used to determine which window is the Listener. This latter restriction arises from the fact that the pointing device returns random positions when the pen is removed or laid on the tablet. If the position were determined simply by the position of the tracking symbol, then characters would be haphazardly given to different windows.

A similar problem arises with mice or pucks when the device is accidentally bumped and the tracking symbol travels into another window. The user might not notice and could thus give the wrong command to the wrong program. Sometimes applications also need to know the tracking symbol position, even when it is outside the window. Therefore, in Sapphire the user must take an explicit action by pressing a button in a window to change the Listener; just moving the tracking symbol is not enough. The press that changes the Listener is not sent to the application program, since this might cause some undesired action.

Another advantage of requiring an explicit press is that the pointing device can then be used in absolute or relative modes by different windows. As was explained above, absolute mode is necessary for digitizing, but relative mode is preferred by most users for other applications. If the current mode is relative, and then a window that uses absolute mode becomes the Listener, the tracking symbol will jump to the absolute location that is specified by the current position of the pointing device. This may well be outside of the current Listener's window, but this causes no problem for Sapphire.

Accelerators. A further important goal of Sapphire is to make the most common window manager operations very easy to specify. A single button press should be sufficient to give the most common commands. On the other hand, we want to provide a large number of different operations to increase the functionality of the system. Many window managers use "pop-up menus" for giving commands. These are small menus that appear when a button is pressed (see Figure 6). The user picks an operation from the menu, and then the menu disappears, restoring the picture that was underneath. The advantage of pop-up menus is that they do not take any screen space when not in use, but a disadvantage is that choosing an item takes a number of steps: first a press to get the menu, then a search to find the correct item in the menu, and finally a selection of that item. Pop-up menus also tend to be computationally expensive, so it may take a noticeable period of time before the menu appears. Experience has shown, with PERQ Sapphire and at Xerox,[16] that experts will be happy to memorize special "accelerators" to avoid using pop-up menus whenever possible. Sapphire provides accelerators, in addition to pop-up menus, in a manner that is easy to learn and remember. The keystroke model[17] suggests that these accelerators will speed up expert performance.

The commands available with a single button press in Sapphire (the accelerators) include top, bottom, moving a window, and changing its size. Another provided operation is making a window full screen. This might be used, for example, when more contextual information is desired during an editing session. The user can define whether "full screen" leaves the icons visible or not. When a window is made full screen, its old position and size are saved so the window can easily be returned to its original place. Similarly, there is a command to move the window entirely off the screen. This can be used to prevent the screen from getting cluttered with windows that are not in use. It also makes the window computations less expensive, since offscreen windows are not considered. A command using the icon brings the window back to its original place on the screen.

Assignment of commands. The title line of a window is divided horizontally into three sections: left, middle, and right. Since title lines are typically about five inches across, each section is about 1½ inches and is therefore easy to hit with the pointing device. The left and right sections have the same functions, since windows are often covered on one side or the other. The most common pointing device for Sapphire has three buttons, so we have 3 (buttons) × 2 (areas: left-right and center) = 6 (functions) available on the title line. The actual assignment of commands to locations can be defined by the user. This is useful for a number of reasons. First, it allows left-handed people to put the major commands under their index finger (when the puck or mouse is moved from the right to the left side, the primary button changes from the index to the ring finger). Another advantage is that novices can define all buttons to provide the same function (e.g., pop-up menu) and thereby avoid confusion until they are comfortable with the system. The best reason for allowing this flexibility, however, is simply that different people want to operate in different ways. For example, some peo-

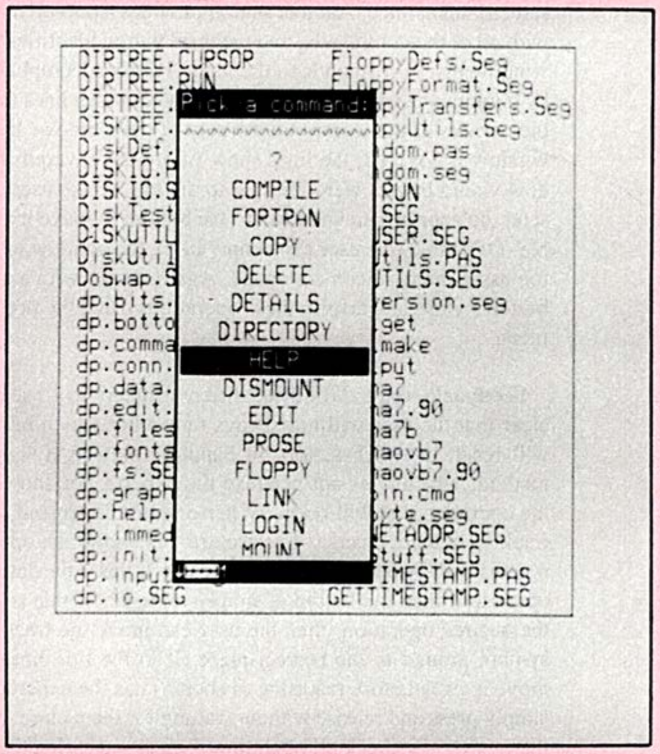
```
DIPTREE.CUPSOP          FloppyDefs.Seg
DIPTREE.RUN             FloppyFormat.Seg
DIPTREE.  Pick a command: ppyTransfers.Seg
DISKDEF.                ppyUtils.Seg
DiskDef.        ?       dom.pas
DISKIO.P                dom.seg
DISKIO.S     COMPILE    RUN
DiskTest     FORTRAN    SEG
DISKUTIL      COPY      USER.SEG
DiskUtil     DELETE     Utils.PAS
DoSwap.S    - DETAILS   UTILS.SEG
dp.bits.    DIRECTORY   ersion.seg
dp.botto                get
dp.comma       HELP     make
dp.conn.                put
dp.data.    DISMOUNT    a7
dp.edit.      EDIT      a7.90
dp.files      PROSE     a7b
dp.fonts     FLOPPY     aovb7
dp.fs.SE      LINK      aovb7.90
dp.graph                in.cmd
dp.help.     LOGIN      yte.seg
dp.immed                ETADDR.SEG
dp.init.      MOUNT     Stuff.SEG
dp.input                TIMESTAMP.PAS
dp.io.SEG               GETTIMESTAMP.SEG
```

Figure 6. A pop-up menu over a background of a directory listing. The command HELP will be executed.

ple would like "top" and "Listener" to happen together, while other people prefer these operations to be performed separately.

Default assignments. The default assignments of operations are as follows: On the ends of the title line are (1) top, (2) bottom, and (3) pop-up menu. The pop-up menu provides all of the commands that can be given directly from the title line, so a novice can always find commands easily. In addition, it allows users with a stylus (one button) to issue all commands. The pop-up menu also includes the command "help" which generates a window in the center of the screen explaining all commands. The pop-up menu contains a number of additional commands, including some for such process control as abort, suspend, and create new shell.

In the center of the title line the more esoteric functions are provided: (4) move/grow, (5) full screen or back from full screen, and (6) offscreen. After selecting "move/grow," the user then selects a position on the window to move or grow from. The corners are grow points, and the sides have both move and grow points. Appropriate feedback shows which operation will be performed, as described below. Since windows in Sapphire may be moved partially offscreen in any direction, it is necessary to be able to move them from any side. It is also useful to be able to grow a window from different points if the user is trying to align one window with other windows. During the move and grow operations, and also during window creation, hairlines are displayed to show the outline for the window so that the user can place it accurately. This feedback also makes it easy to identify what the system expects the user to do.

The icons are too small to have three areas, so we are limited to three functions total (one for each button). One button performs "top" and "Listener" and "back from offscreen" all at once. Another button provides a pop-up menu with all of the commands, and the third button identifies the window that corresponds to the icon. This is accomplished by blinking the window and the icon, and drawing lines from the corners of the icon to the corners of the window. If the window is covered, the lines show where the covered window would be if it were brought to the top, so the user can send the appropriate windows to the bottom to make it visible. Of course, the user can simply bring the window to the top using the first icon command. Again, a user with a one-button stylus can perform these operations using the pop-up menu.

Feedback. With all of these different functions, it seems clear that the user will not always remember which button will result in which action. In Sapphire there is a simple method, which does not penalize the experts, for showing the operation that will occur. When a button is pressed, the tracking symbol changes to a picture that shows the operation that will be performed (see Figure 7). If this is the desired operation, then the button is simply released. If this is *not* the desired operation, then the user can move the tracking symbol around to the correct place (if in the title line) or move it away before releasing to abort. Thus the expert can simply press and release without waiting for the picture, and the novice can check all operations before executing them. Some systems provide an "undo" command, so novices can execute a command and then undo it if it was the wrong one. Users prefer knowing in advance what command they will execute, however, so Sapphire provides feedback before commands are executed. All commands can easily be reversed or aborted in Sapphire, so a special "undo" is not needed.

While performing such multistep tasks as growing a window or choosing from a pop-up menu, Sapphire also displays an appropriate tracking symbol picture. This can be used, for example, to verify whether a "move" or a "grow" will be performed. The tracking symbol picture unambiguously shows the user what to do, which reduces the confusion and difficulty. Also, the user can always abort these operations by hitting a keyboard key, so the user is never stuck in a mode without knowing how to leave.

The default tracking-symbol pictures that are used were drawn by the author and a graphic artist (see Figure 7), but they can easily be changed. For example, other versions of the pictures are shown in Aaron Marcus's article on p. 31.

Keyboard commands. For some reason there are a number of people who prefer not to use a pointing device. Therefore, all commands in Sapphire are also available from the keyboard. Since there are a number of commands, we use a special prefix key (that does not have a standard ASCII interpretation). This avoids the problem of taking many keys away from application programs. The keyboard commands are also useful if the pointing device is broken, or if some process has reserved it or changed the tracking in arbitrary ways. For example, Sapphire allows application programs to specify that the tracking should be on a grid or restricted within a specified box or turned off altogether. This makes the operating system more efficient, since the application program does not have to wait in a busy-loop, monitoring the pointing device. However, since the prefix key (which can be changed by users) returns the tracking to the standard default, we have ensured that users can always give commands either from the keyboard or the pointing device.

Most keyboard commands operate on the current Listener. Some commands, however, are global. For example, one of the keys on the PERQ keyboard is labeled "Help," and this provides help for Sapphire if typed after the prefix key. Another global command displays the icon window if it has been moved offscreen or covered.

To designate different windows as the Listener using the keyboard, there are a pair of commands that change the Listener to another window in a certain order. Currently, there is a ring of windows ordered temporally by the time they were last the Listener. The window that was the last Listener is previous to the current Listener, with the window previous to that before it (see Figure 8). Thus, you can go to the previous Listener with one command. If the user is operating on a pair of windows, for example, this makes it easy to switch back and forth. When new windows are created, they are put at the "end" of the ring, which is just *after* the current Listener (see Figure 8). Thus the user can change from the current Listener to the most recently created window with one command; just move *forward* in the Listener ring. This will be useful when the user causes a new window to be created and then wants to operate on this window.

Although this order is very appealing, it has a serious problem. When the Listener is changed from the current Listener

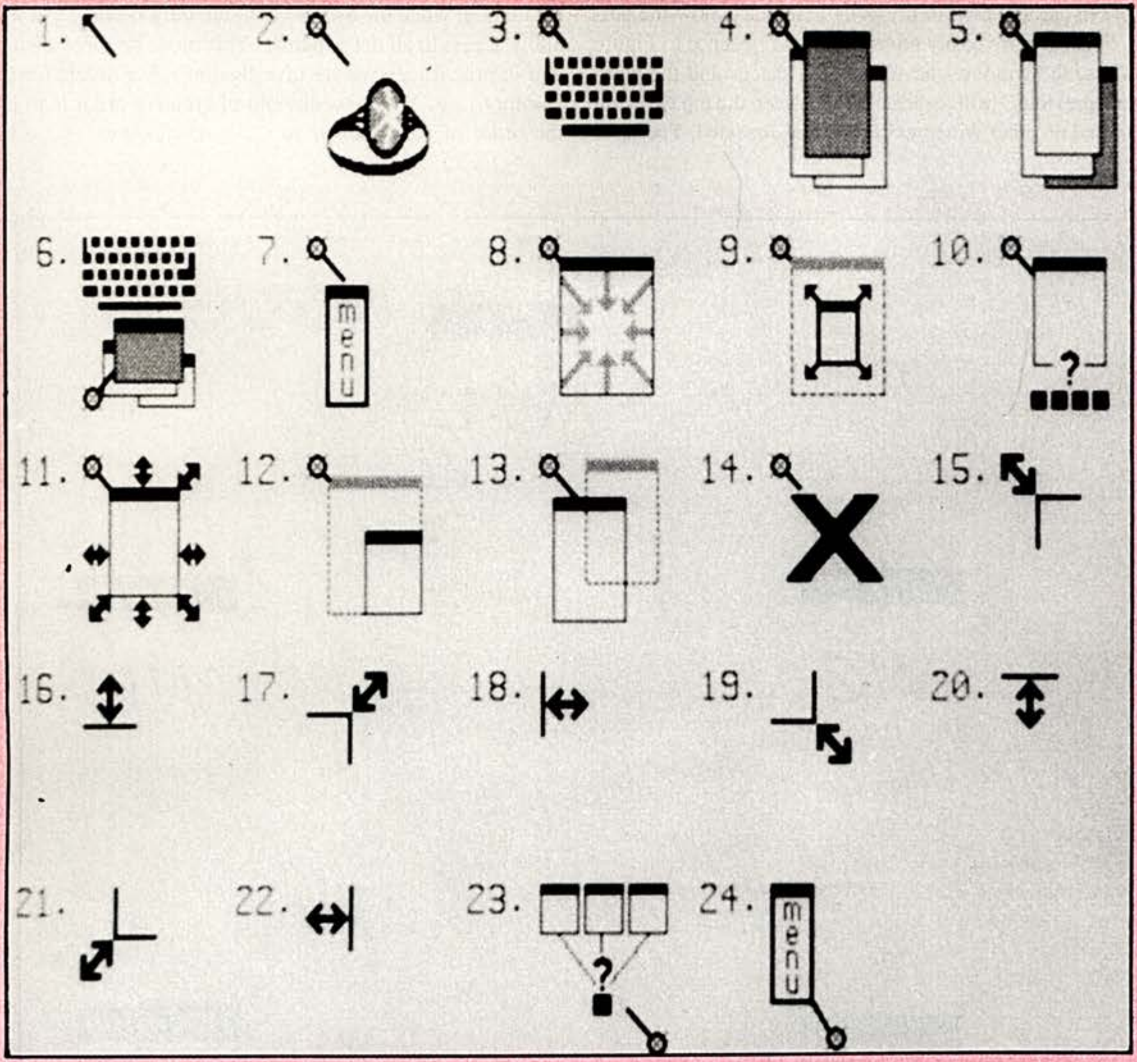

Figure 7. Tracking symbols used in Sapphire. When the user presses down on the pointing device, the tracking symbol picture changes to show what operation will be performed. If the operation requires multiple actions (for example, when growing a window), the picture changes at each step to show what is expected next. The pictures above represent:

1. The system default cursor.
2. The default Sapphire cursor (a star Sapphire ring).
3. Make this window be the Listener.
4. Bring this window to the top.
5. Send this window to the bottom.
6. Top and Listener (used in the icon).
7. Get a pop-up menu of commands.
8. Return the window from full screen to original position.
9. Make the window full screen.
10. Identify the icon for this window.
11. Change size or position of window from any side or corner.
12. Change the window's size.
13. Change the window's position.
14. User-specified abort of any operation.

15-22. Specify corner or side of the window.

23. Identify the window for this icon.
24. Get a pop-up menu from icon.

to the previous Listener, the old current Listener becomes the previous, and the old previous Listener is now the current. (This is more easily understood by reference to Figure 8.) Thus, the windows have switched places and the command "previous" will just oscillate between the top two windows, and no other windows can ever be accessed. The same problem occurs for moving forward. Therefore, there must be a mode, when the user is changing the Listener, that will allow access to all the windows. This mode has been a problem in practice, so we are investigating other orders for the Listener ring. The most obvious alternative order is to use the order of the windows in the icon window.

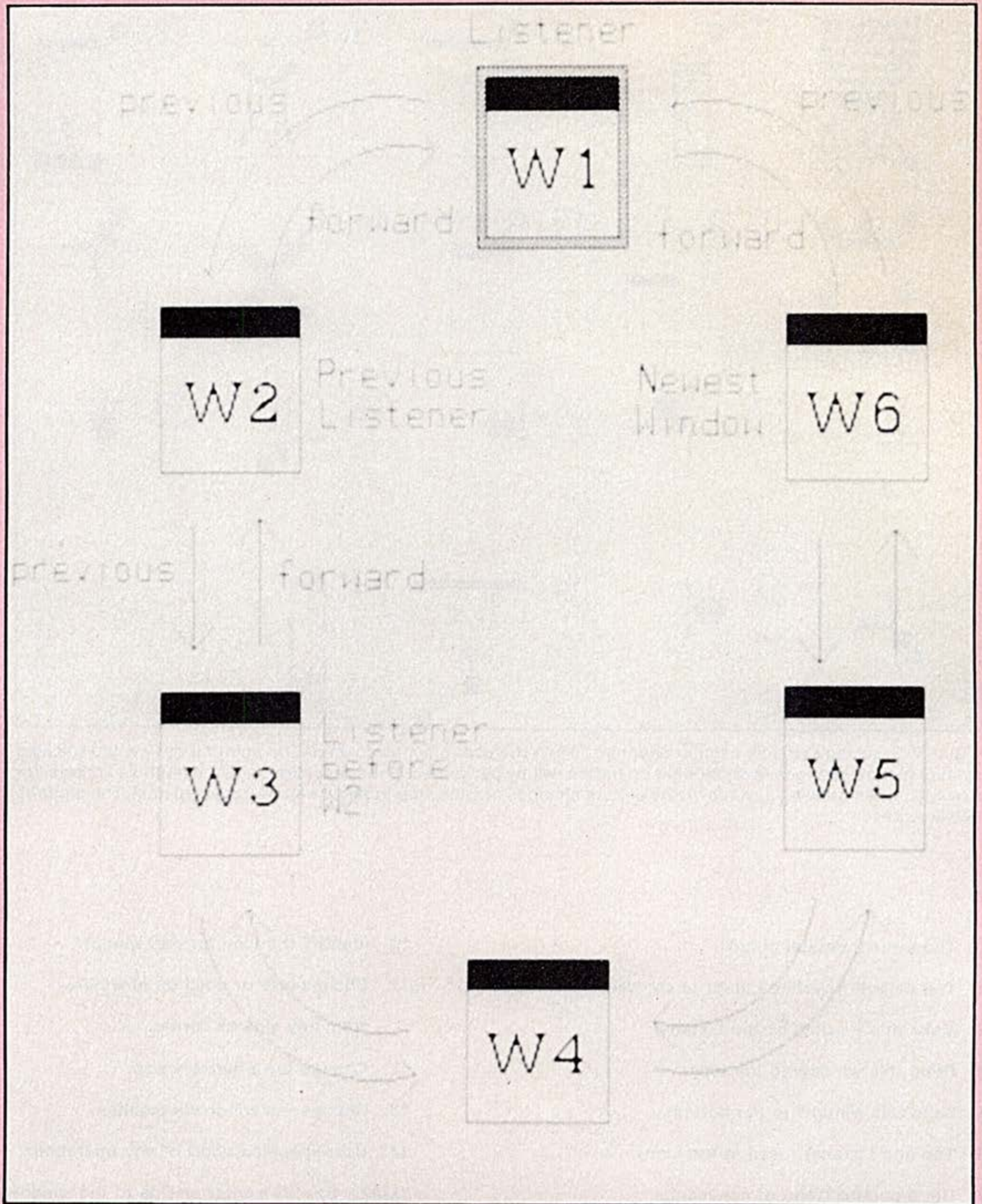

Figure 8. The Listener ring. W1 is the current Listener. The previous Listener was W2, so giving the "previous-Listener" command will move from W1 to W2. Listener before W2 was W3, etc. Window W6 was newly created and has never been Listener so it is at the "end" of the ring. The command "forward Listener" from W1 will get to W6. When windows are created, they are added "forward" from the current Listener (e.g., between W1 and W6 in this case). If W4 is made the current Listener, then it will be removed from the ring (so forward from W5 is W3, and previous to W3 is W5), and then added between W1 and W6. When using the "previous-Listener" command to move around the ring, a mode is needed to prevent W2 and W1 from just repeatedly swapping places.

Conclusions

Sapphire incorporates a number of innovations in window management and user interface design. Icons in Sapphire are used in a novel way to enhance the user's productivity when doing several tasks concurrently. Eight pieces of status information are shown in the icon, so the user can easily tell whether the process running in the window has gotten an error or wants attention and what percentage of the task has been completed. This allows the user to monitor and control multiple processes more easily.

The user interface of Sapphire provides full functionality from both the pointing device and the keyboard and is easy for the novice while providing simple and powerful operations for experts. All commands are available from pop-up menus, but accelerators allow the most common commands to be executed with a single button press. The picture in the tracking symbol changes to show the operation that will be performed. This user interface promotes experimentation, since there is always appropriate feedback, and it is always possible to abort an operation once it has been started. This flexibility and power has been provided in a way that does not restrict application programs to a particular style of interaction or a predefined analogic view of the system.

Sapphire is currently in use by a growing community of people in different fields. Even though some of its features are not yet extensively used, the overall consensus is enthusiastically positive. Sapphire is continually being modified based on user feedback, as we discover how to make it even easier to use.■

Acknowledgments

The design of Sapphire grew out of discussions with Gene Ball, Fred Hansen, Dave Golub, and many others at Carnegie-Mellon University and PERQ Systems Corporation. Amy Butler and David Golub have been largely responsible for maintaining Sapphire. For help and support with this article, I would like to thank my wife, Bernita Myers, and Alain Fournier, William Buxton, Brian Rosen, Joyce Swaney, and many others at the University of Toronto and PERQ Systems Corporation.

References

1. Brian Rosen, "PERQ: A Commercially Available Personal Scientific Computer," *IEEE Compcon Digest*, Spring 1980, pp 484-485.
2. R. Rashid and G. Robertson, "Accent: A Communication Oriented Network Operating System Kernel," *Proc. Eighth Symp. Operating Systems Principles*, Dec. 1981, Asilomar, Calif., pp. 64-75.
3. David Canfield Smith, Charles Irby, Ralph Kimball, Bill Verplank, and Erik Harslem. "Designing the Star User Interface," *Byte Magazine*, Apr. 1982, pp. 242-282.
4. Brad A. Myers, "A Complete Implementation of Covered Windows for a Heterogeneous Environment," Dynamic Graphics Project, tech. memo, U. of Toronto Comp. Sci. Dept., 1984.
5. D. M. Richie and K. Thompson, "The UNIX Time-sharing system," *Comm. ACM,* Vol. 17, No. 7, July 1974, pp. 365-375.
6. Richard J. Beach, "Experience with the Cedar Programming Environment for Computer Graphics Research," *Proc. Graphics Interface '84,* May 28-June 1, 1984, Ottawa, Ontario, Canada, pp. 65-79.
7. Warren Teitelman, *A Display Oriented Programmer's Assistant*, Palo Alto: Xerox PARC CSL-77-3, Mar, 8, 1977.
8. Larry Tesler, "The Smalltalk Environment," *Byte Magazine*, Aug. 1981, pp. 90-147.
9. D. Weinreb and D. Moon, *Introduction to Using the Window System*, Symbolics, Inc., 1981.
10. Rob Pike, "Graphics in Overlapping Bitmap Layers," *ACM Trans. Graphics,* Vol. 2, No. 2, Apr. 1983, pp. 135-160.
11. W.K. English, D.C. Engelbart, and M.L. Berman, "Display Selection Techniques for Text Manipulation," *IEEE Trans. Human Factors in Electronics,* Vol HFE-8, No. 1, Mar. 1967.
12. Gregg Williams, "The Lisa Computer System," *Byte Magazine,* Feb. 1983, pp. 33-50.
13. Frank Halasz and Thomas P. Moran, "Analogy Considered Harmful," *Proc. Human Factors in Computer Systems Conf.*, Mar. 15-17, 1982, pp. 383-386.
14. Brad A. Myers, "The Importance of Percent-Done Progress Indicators for Computer-Human Interfaces," Dynamic Graphics Project, tech. memo, University of Toronto Comp. Sci. Dept., 1984.
15. "PERQ POS Operating System Manual," *PERQ System Software Reference Manual, POS Version G3,* PERQ Systems Corporation, Pittsburgh, Pa. May 1983.
16. Theresa Roberts, private conversation about experience at Xerox SDD on the Star development environment, Mar. 1984.
17. S. K. Card, T. P. Moran, and A. Newell, "The Keystroke-Level Model for User Performance Time with Interactive Systems," *Comm. ACM,* Vol. 23, No. 7, July 1980, pp. 396-410.

Brad A. Myers is a PhD candidate in computer science at the University of Toronto. From 1980 until 1983 he worked at PERQ Systems Corporation where he designed and implemented the Sapphire window manager and numerous PERQ demonstrations for the Siggraph equipment exhibition. His research interests include user interfaces, interaction techniques, window management, programming environments, debugging, and graphics. Myers received the BS and MS degrees from the Massachusetts Institute of Technology. While attending MIT, he was a research intern at Xerox PARC. He is a member of Siggraph, ACM, the Canadian Information Processing Society, and the IEEE Computer Society.

Myers's address is Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada, M5S 1A4.

PERQ's address is PERQ Systems Corporation, 2600 Liberty Avenue, P.O. Box 2600, Pittsburgh, PA, USA, 15230.