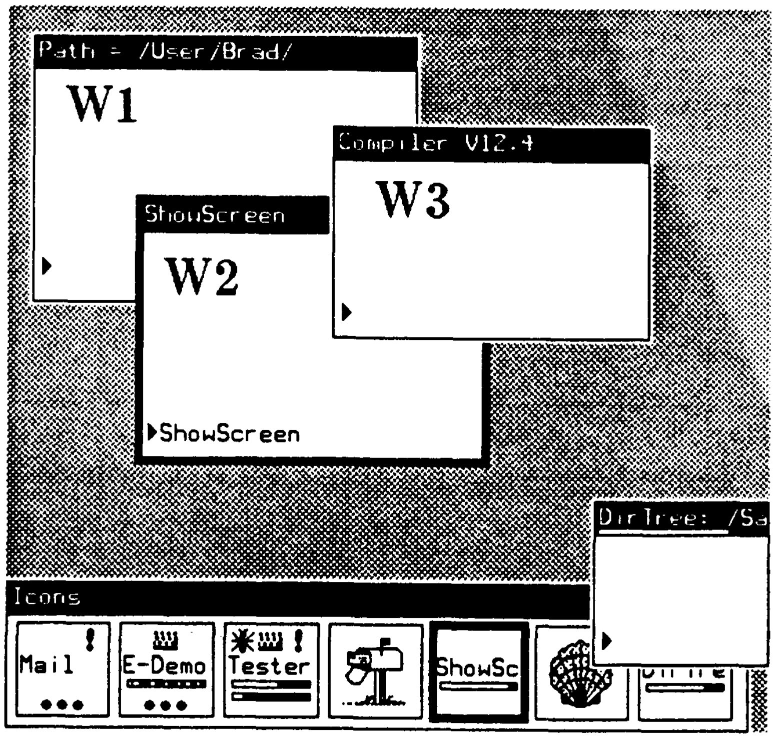
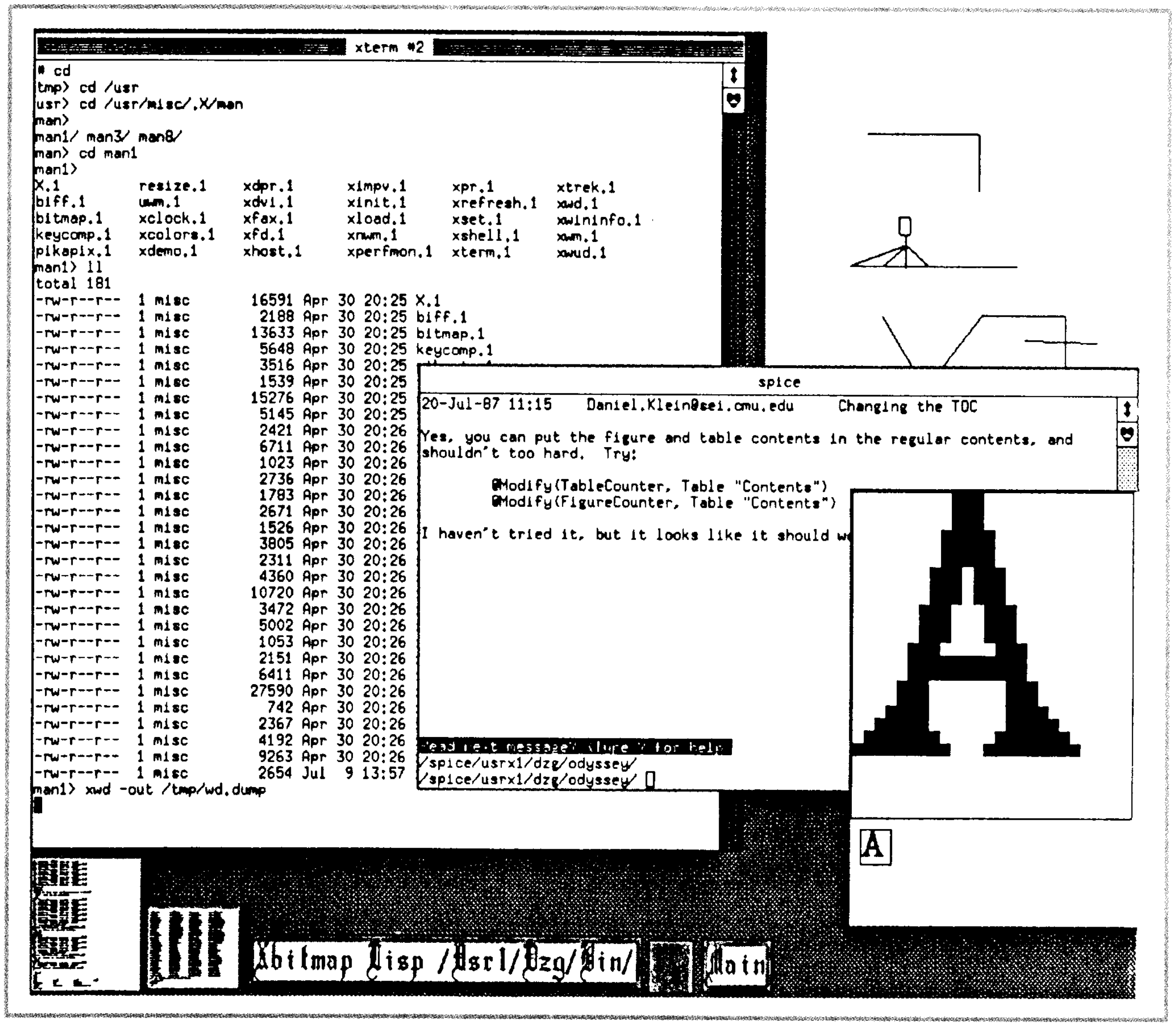
# A Taxonomy of Window Manager User Interfaces

Brad A. Myers
Carnegie Mellon University



This article presents a taxonomy for the user-visible parts of window managers. It is interesting that there are actually very few significant differences, and the differences can be classified in a taxonomy with fairly limited branching. This taxonomy should be useful in evaluating the similarities and differences of various window managers, and it will also serve as a guide for the issues that need to be addressed by designers of future window manager user interfaces. The advantages and disadvantages of the various options are also presented. Since many modern window managers allow the user interface to be customized to a large degree, it is important to study the choices available.

A window manager is a software package that helps the user monitor and control different contexts by separating them physically onto different parts of one or more display screens. At its simplest, a window manager provides many separate terminals on the same screen, each with its own connection to a time-sharing computer. At its most advanced, a window manager supports many different activities, each of which uses many windows, and each window, in turn, can contain many different kinds of information including text, graphics, and even video. Window managers are sometimes implemented as part of a computer's operating system and sometimes as a server that can be used if desired. They



```
# cd
tmp> cd /usr
usr> cd /usr/misc/.X/man
man>
man1/ man3/ man8/
man> cd man1
man1>
X.1          resize.1     xdpr.1       ximpv.1      xpr.1        xtrek.1
biff.1       uwm.1        xdvi.1       xinit.1      xrefresh.1   xwd.1
bitmap.1     xclock.1     xfax.1       xload.1      xset.1       xwininfo.1
keycomp.1    xcolors.1    xfd.1        xnwm.1       xshell.1     xwm.1
pikapix.1    xdemo.1      xhost.1      xperfmon.1   xterm.1      xwud.1
man1> ll
total 181
-rw-r--r--  1 misc        16591 Apr 30 20:25 X.1
-rw-r--r--  1 misc         2188 Apr 30 20:25 biff.1
-rw-r--r--  1 misc        13633 Apr 30 20:25 bitmap.1
-rw-r--r--  1 misc         5648 Apr 30 20:25 keycomp.1
-rw-r--r--  1 misc         3516 Apr 30 20:25
-rw-r--r--  1 misc         1539 Apr 30 20:25
-rw-r--r--  1 misc        15276 Apr 30 20:25
-rw-r--r--  1 misc         5145 Apr 30 20:25
-rw-r--r--  1 misc         2421 Apr 30 20:26
-rw-r--r--  1 misc         6711 Apr 30 20:26
-rw-r--r--  1 misc         1023 Apr 30 20:26
-rw-r--r--  1 misc         2736 Apr 30 20:26
-rw-r--r--  1 misc         1783 Apr 30 20:26
-rw-r--r--  1 misc         2671 Apr 30 20:26
-rw-r--r--  1 misc         1526 Apr 30 20:26
-rw-r--r--  1 misc         3805 Apr 30 20:26
-rw-r--r--  1 misc         2311 Apr 30 20:26
-rw-r--r--  1 misc         4360 Apr 30 20:26
-rw-r--r--  1 misc        10720 Apr 30 20:26
-rw-r--r--  1 misc         3472 Apr 30 20:26
-rw-r--r--  1 misc         5002 Apr 30 20:26
-rw-r--r--  1 misc         1053 Apr 30 20:26
-rw-r--r--  1 misc         2151 Apr 30 20:26
-rw-r--r--  1 misc         6411 Apr 30 20:26
-rw-r--r--  1 misc        27590 Apr 30 20:26
-rw-r--r--  1 misc          742 Apr 30 20:26
-rw-r--r--  1 misc         2367 Apr 30 20:26
-rw-r--r--  1 misc         4192 Apr 30 20:26
-rw-r--r--  1 misc         9263 Apr 30 20:26
-rw-r--r--  1 misc         2654 Jul  9 13:57
man1> xwd -out /tmp/wd.dump
```

```
20-Jul-87 11:15    Daniel.Klein@sei.cmu.edu     Changing the TOC

Yes, you can put the figure and table contents in the regular contents, and
shouldn't too hard.  Try:

        @Modify(TableCounter, Table "Contents")
        @Modify(FigureCounter, Table "Contents")

I haven't tried it, but it looks like it should w

Read next message? (type ? for help
/spice/usrx1/dzg/odyssey/
/spice/usrx1/dzg/odyssey/
```

**Figure 1. An example of a typical screen using the X window manager[4] with overlapping windows. Some windows have title lines (the top-left window's says "xterm #2"). The background, where there are no windows, is gray. The small windows at the bottom are icons.**

can even be implemented by individual application programs or programming environments.

Window managers have become popular primarily because they allow separate activities to be put in physically separate parts of the computer screen. The user of a computer is frequently shifting focus from one activity to another, including such small shifts as changing from editing one file in a text editor to editing another, and such large context shifts as changing from compiling a program to reading mail.

Before window managers, people had to remember their various activities and how to switch back and forth. Window managers allow each activity to have its own separate area of the screen (its own "window"). Switching from one window to another is usually very simple. This physical separation is even more important when the operating system allows multiple activities to operate at the same time ("multiprocessing"). For example, in Unix, the user can compile one file at the same time a different file is being edited. On a conventional terminal, if the compiler process outputs any data, it is confusingly interspersed with the editor's display. If the two processes request input at the same time, the user may give the input to the wrong program. Window managers help with these problems by providing separate areas in which each process can perform input and output.

Another advantage of window managers is that they provide a higher level interface to the mouse, keyboard, and screen, and therefore can support much higher quality user interfaces. For example, the window managers on the Star[1,2] and Macintosh[3] help support the metaphor that using the computer is like doing operations on a physical desk. This higher level interface can also make application code more portable from one machine to another, since the same window manager procedural interface can be provided on different machines. This was an important motivation for the development of the X window manager.[4]

Today there are a large number of window managers in existence from many companies and research groups, and more are being created all the time. In surveying these window managers, it became clear that there are many similarities between all of them, and the differences can be characterized on a small number of different axes. (This survey was started at the Alvey MMI Workshop on Window Management.[5]) Most of the ideas seem to have originated at the Xerox Palo Alto Research Center, including windows in general (Smalltalk[6]), pop-up menus,[7] icons (Tajo[8,9] and Star[1,2]), and tiled windows (Cedar[10,11]). Of course every window manager has its own original aspects, but most of the important features of the user interfaces of window managers do not seem to vary markedly.

With the advent of the X window manager,[4] which is rapidly becoming a de facto standard, the study of the user interface component is becoming more critical. This is because X and many other modern window managers allow the user interface to be changed, while still maintaining the same application interface. User interface designers therefore are faced with not only a choice for the user interface of their application, but also for that of the window manager. It is therefore important to focus on the different choices in the user interface component of window managers. This article presents a taxonomy of the choices used in existing window manager user interfaces, along with some advantages and disadvantages of each choice.

(At the time of this writing, Xerox, AT&T, and Sun had just announced a portable window-manager user interface called "Open Look," which apparently will be implemented on multiple-window managers, including X [4] and NeWS.[12] Open Look, which is based partially on the user interface of the Xerox Star, is designed to match the ease of use of the Macintosh, and thereby make Unix systems more user friendly.)

## Definition of terms

The previous section defined "window managers" and discussed the reason they are so popular. This section defines some related terms that are important for understanding how window managers work.

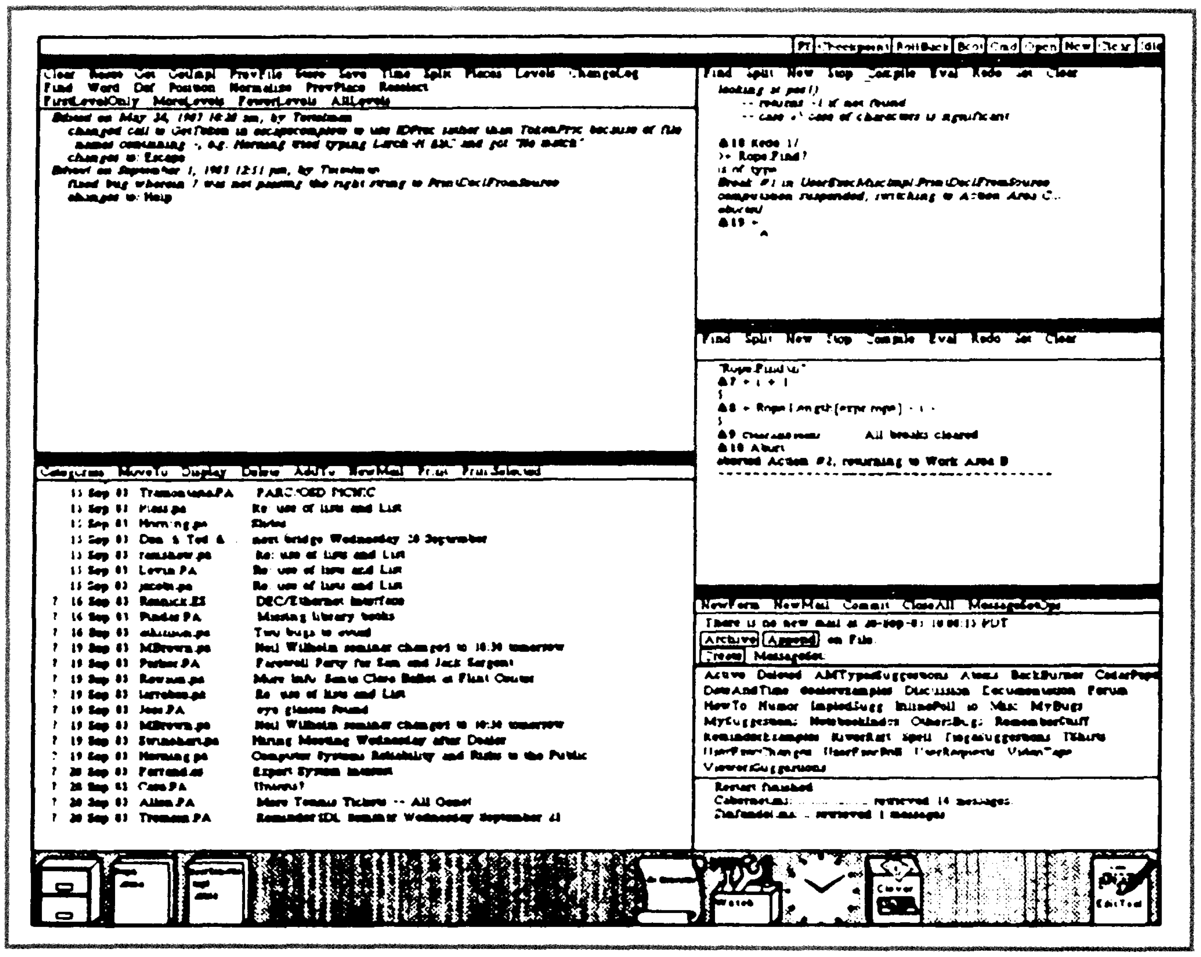
A window manager can be logically divided into two layers, each of which has two parts. The base layer implements the basic functionality of the window manager. The two parts of this layer handle the display of graphics in windows and access to the various input devices (usually a keyboard and a pointing device such as a mouse). The primary interface of this layer is to other programs, and it is called the window manager's *application* or *program interface*. The base layer is not discussed further in this article. The other layer of window managers is the *user interface*. This includes all aspects that are visible to the user. Sometimes the base layer is called a *window system*, reserving the name "window manager" for the user interface layer. Since this article deals only with the user interface layer, the term "window manager" is used here.

The two parts of the user interface layer are the *presentation*, which is composed of the pictures that the window manager displays, and the *operations*, which are the commands the user can give to manipulate the windows and their contents. Figure 1 shows windows that demonstrate different aspects of the presentation, including patterns or pictures for the area where there are no windows, title lines and borders for windows, etc. Examples of the operations that may be provided for windows include moving them around on the screen and specifying their size.

One very important aspect of the presentation of windows is whether they can *overlap* or not. Overlapping windows, sometimes called *covered* windows, are a feature allowing a window to be partially or totally on top of another window, as shown in Figure 1. This is also sometimes called the *desktop metaphor*, since windows can cover each other like pieces of paper can cover each other on a desk. (There are usually other aspects to the desktop metaphor, however, such as presenting file operations in a way that mimics office operations, as in the Star office workstation.[1,2]) The other alternative is called *tiled* windows, which means that windows are not allowed to cover each other. Figure 2 shows an example of tiled windows. The advantages and disadvantages of each are discussed below. Obviously, a window manager that supports covered windows can also allow them to be side by side, but not vice versa. Therefore, a window manager is classified as covered if it allows windows to overlap.

Another important aspect of the presentation of windows is the use of *icons*. These are small pictures that represent windows. They are used because there would otherwise be too many windows to conveniently fit on the screen and manage easily. When a window is not in use, it can be removed and replaced with its icon, and later conveniently retrieved when needed. Figure 3 shows examples of icons from some different window managers. The section on icons discusses the options available for icons in more detail.

An important aspect of window managers is how the user changes which window is connected to the key-



PT Checkpoint RollBack Boot Cmd Open New Clear Idle
Clear Reset Get GetImpl PrevFile Store Save Time Split Places Levels ChangeLog
Find Word Def Position Normalize PrevPlace Reselect
FirstLevelOnly MoreLevels FewerLevels AllLevels
Find Split New Stop Compile Eval Redo Set Clear
Find Split New Stop Compile Eval Redo Set Clear
Categories MoveTo Display Delete AddTo NewMail Print PrintSelected
NewForm NewMail Commit CloseAll MessageSetOps
Archive Append
Create
Watch
Clover
EditTool

```
Edited on May 24, 1983 10:20 am, by Teitelman
   changed call to GetToken in escapecomplete to use IDProc rather than TokenProc because of file
    names containing -, e.g. Horning tried typing Larch -H &XL' and got "No match"
   changes to: Escape
Edited on September 1, 1983 12:51 pm, by Teitelman
   fixed bug whereas I was not passing the right string to PrintDeclFromSource
   changes to: Help
```

```
looking at pos[]
    -- returns -1 if not found
    -- case ∈ case of characters is significant

&18 Redo 1/
>+ Rope.Find?
is of type
Break #1 in UserExecMiscImpl.PrintDeclFromSource
computation suspended, switching to Action Area C...
aborted
&19 ←
```

```
"Rope.Find(s..."
&7 ← i + 1
$
&8 ← Rope.Length[expr.rope] - i -
$
&9 ClearAllBreaks          All breaks cleared
&10 Abort
aborted Action #2, returning to Work Area B
-----------------------------------------------
```

```
   13 Sep 83  TramontanaPA      PARC-OSD PICNIC
   13 Sep 83  Plass.pa          Re: use of lists and List
   13 Sep 83  Horning.pa        Slides
   13 Sep 83  Dan & Ted &...    next bridge Wednesday 20 September
   13 Sep 83  ramshaw.pa        Re: use of lists and List
   13 Sep 83  Levin.PA          Re: use of lists and List
   15 Sep 83  jacobi.pa         Re: use of lists and List
?  16 Sep 83  Rennick.ES        DEC/Ethernet interface
?  16 Sep 83  Pandas.PA         Missing library books
?  16 Sep 83  atkinson.pa       Two bugs to avoid
?  19 Sep 83  MBrown.pa         Neil Wilhelm seminar changed to 10:30 tomorrow
?  19 Sep 83  Parker.PA         Farewell Party for Sam and Jack Sargent
?  19 Sep 83  Rowson.pa         More info. Santa Clara Ballet at Flint Center
?  19 Sep 83  larrabee.pa       Re: use of lists and List
?  19 Sep 83  Jean.PA           eye glasses found
?  19 Sep 83  MBrown.pa         Neil Wilhelm seminar changed to 10:30 tomorrow
?  19 Sep 83  Swinehart.pa      Hiring Meeting Wednesday after Dealer
?  19 Sep 83  Horning.pa        Computer Systems Reliability and Risks to the Public
?  20 Sep 83  Ferrand.es        Expert System manual
?  20 Sep 83  Cate.PA           Unsent?
?  20 Sep 83  Allen.PA          More Tennis Tickets -- All Gone!
?  20 Sep 83  Treman.PA         Reminder:IDL seminar Wednesday September 21
```

```
There is no new mail at 20-Sep-83 10:00:15 PDT
Archive Append on File:
Create MessageSet:
Active Deleted AMTypeSuggestions Atoms BackBurner CedarPapers
DateAndTime DesktopExamples Discussion Documentation Forum
HowTo Humor ImpledSugg InitiatePoll io Misc MyBugs
MySuggestions Notebookideas OtherBugs RememberStuff
RemindersExamples RiverRat Spell TiogaSuggestions TXhure
UserPaneChanges UserProfile UserRequests VideoTape
ViewerSuggestions
  Restart finished
  Cabernet.ms: ........ retrieved 14 messages.
  Chefanelet.ms ... retrieved 1 messages.
```

Figure 2. A screen from the Cedar[10,11] window manager. Windows are "tiled" into two columns. There is a row of icons along the bottom. Each window has a fixed menu of commands below the title line.

board. Although there will typically be multiple windows, there is usually only one keyboard for each user. Therefore, only one window at a time can be attached to the keyboard. This window is termed the *listener*, since it is listening to the user's typing. Another term for this window is the *input* (or *keyboard*) *focus*. Older systems called the listener the "active window" or "current window," but these are poor terms, since in a multiprocessing system, many windows can be actively outputting information at the same time. Window managers provide various ways to specify and show which window is the listener.

Most window managers use some form of pointer, which is an input device that returns a 2D value used to identify locations on the screen. Pointing devices are typically used for specifying window size and position, for selecting 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 last use is called *digitizing*). Examples of pointing devices are light pens, electromagnetic tablets with pucks or pen-like styli, touch-sensitive surfaces (touch tablets or touch screens), trackballs, and mechanical or optical mice.[14] Since the most popular pointing device for window managers is a mouse, the term *mouse* will often be used in this article to mean *pointing device*.

Light pens and touch screens are used for pointing directly at the screen, but with the other types the user moves a device on the desk or on a special surface, and a small picture, called the *tracking symbol* or *cursor*, follows the movement on the screen. In many window managers the picture for the cursor can be changed, but a common picture is an arrow pointing to the upper left.

Pointing devices usually have one or more buttons. For example, there are typically one to three buttons on the top of a mouse. Some window managers allow the user

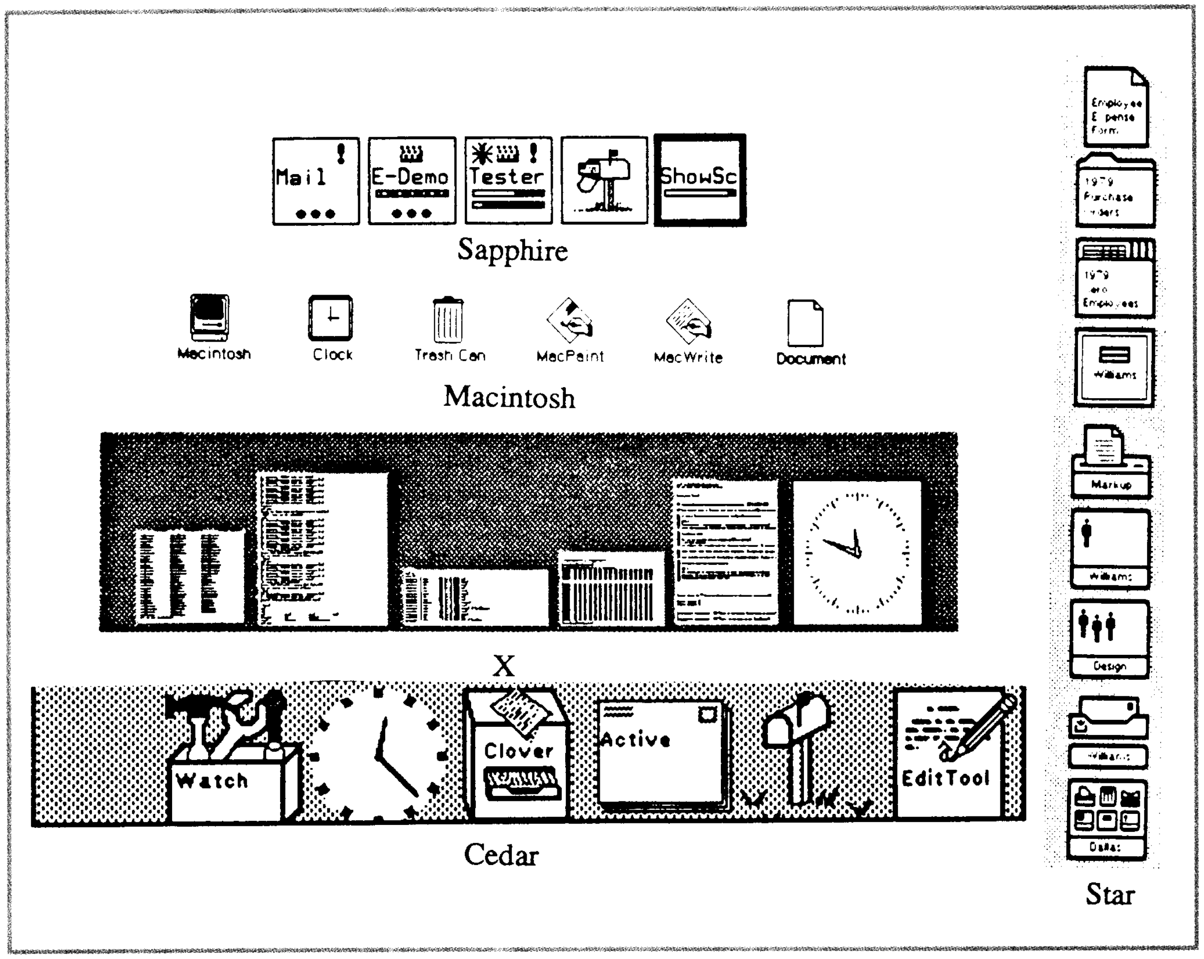


**Figure 3. Examples of icons from different systems: Sapphire,[13] Macintosh,[3] X,[4] Cedar,[10] and Star.[1] Some of the X icons contain the actual text displayed in the window in a tiny (unreadable) font.**

to press two or three times quickly to specify additional commands. This is called *multiclicking* (for example, pressing twice quickly is double-clicking). Window managers may also support holding down keyboard keys (such as the shift key) while pressing a mouse button. This is often used to modify the button's meaning.

## The window manager versus add-ons

To compare window managers, it is first necessary to establish the boundaries of discussion. A window manager provides the basic service of managing different windows on the screen, as defined above. In many systems, however, other services are also provided, and these are often classified as part of the window manager. By providing these services in a central place, the system promotes consistency and makes applications easier. To compare the window managers of these different systems, however, it is important to classify which aspects are being compared and which are considered add-on services. This section discusses some of these add-ons so that the rest of the article can concentrate on the window manager portion itself.

Some common add-ons are the following:

1. a typescript package (handles user typing)
2. entire editors
3. a graphics package for output (also called the *imaging model*
4. menus of various kinds
5. forms (also called *dialogue boxes*)
6. scrolling mechanisms
7. general tool kits (which usually include menus, forms, and scrolling mechanisms)

**Table. Window managers discussed in this article.**

| Name | Created by | Primary Computer | Tiled or Covered | Comments | References |
|---|---|---|---|---|---|
| Smalltalk | Xerox PARC | Alto | C | first window system | 6,7,29,30 |
| DLisp | Xerox PARC | Alto+mainframe | C | | 40 |
| Interlisp-D | Xerox PARC | Dorado | C | | 35 |
| Tajo | Xerox SDD | Dandelion | C | first use of icons, renamed "XDE" | 8,9,38 |
| Star | Xerox SDD | Dandelion | T | first product with windows | 1,2,46 |
| Blit | AT&T Bell Labs | Blit | C | 1st documented impl.; terminal emulator | 36 |
| Display Manager | Apollo | Apollo | C | originally had no mouse | 49 |
| ?? | Symbolics | Symbolics | C | uses multi-clicking | 48 |
| Sapphire | Three Rivers | PERQ | C | active icons | 13 |
| PNX | ICL | PERQ | C | feedback is full windows | 41 |
| SunWindows | Sun | Sun | C | | 17 |
| Cedar | Xerox PARC | Dorado | T | first tiled; used graphics package | 10,11,20 |
| Window Manager | Apple | Lisa | C | | 18 |
| Window Manager | Apple | Macintosh | C | popularized windows & mouse | 3,19 |
| Andrew | CMU ITC (IBM) | IBM-RT, Sun | T | | 16 |
| Whitechapel | Whitechapel | MG-1 | C | | 23 |
| RTL/CRTL | Siemens | PERQ | T | no columns; used constraints | 33 |
| MSWindows | Microsoft | IBM-PC | T | current version supports covered also | 28 |
| Viewpoint | Xerox SDD | 6085/1186 | T or C | successor to Xerox Star | 32 |
| X | MIT Project Athena | <many> | C | first "portable" WM; emerging standard | 4,24,27 |
| NeWS | Sun | Sun, etc. | C | uses Postscript; *nee* "Sundew" | 12,21 |

8. user interface management systems or UIMSs[15] (which usually include a tool kit)

## Typescript package

An example of a service that is often provided by window managers is the handling of typing. This is often called a *typescript package*, and it usually supports some rudimentary line editing (backspace, delete line, etc.). The idea is to mimic the teletype interface to terminals provided by conventional time-sharing operating systems. Most programming languages (for example C and Pascal) have as a function to read a line of text. When this function is executed, the user is typically allowed to edit the typed line before using carriage-return to confirm the entry. The typescript package handles this input also. In addition, it may provide more elaborate commands, and, in the extreme, be a full-fledged editor, as in the Andrew system.[16] In a window system, the typescript package may also provide the ability to copy text from one window to another, as in SunWindows.[17] An advanced form of this copying is the clipboard in the Lisa [18] and Macintosh,[3] which provides the ability to copy arbitrary text and graphics from one window to another.

## Editors

In addition to the typescript package used to handle command typing, some window managers include an entire text editor, which can be used for preparing documents and programs. This may or may not be the same package used for handling typing to programs.

## Graphics package

Some window managers provide a sophisticated graphics package for application programs to help them produce output. Clearly, the window manager needs to output some graphics to draw the title lines, window borders, icons, backgrounds, etc. The primitives that the window manager provides for handling output is called the "imaging model" of the window manager.

Some window managers, such as SunWindows and X, provide a simple imaging model and expect that more sophisticated graphics packages will be built using the window manager. This allows more flexibility, since multiple graphics packages can be used. For example, the CORE, GKS, and PHIGS graphics packages have all been implemented on top of SunWindows. In addition, the graphics operations may be more efficient, since the window manager can export the primitives supported by the hardware. The interface to the window manager may be simpler, since there are fewer primitives.

Other window managers are built on top of powerful graphics packages. For example, the Macintosh is on top of QuickDraw,[19] Cedar is on top of CedarGraphics,[20] and NeWS,[12] which was originally called SunDew,[21] is on top of a version of Postscript.[22] Adobe Systems and Next are creating another version of Postscript called

Display Postscript to serve as the imaging model for future window managers. The advantage of using an underlying graphics package is that the window manager can provide a more attractive presentation. For example, the Macintosh window manager displays drop-shadows and rounded corners. Other advantages include a more consistent interface to and between applications and better device independence.

### User interface tool kits

Another service often provided by window managers is a library of procedures to help applications create their user interfaces. For example, almost every window manager provides a menu package. Whitechapel[23] also provides scroll bars that can be displayed on the windows. The Macintosh comes with a complete "Toolbox,"[19] including menus, dialogue boxes, scroll bars, and text-editing. A full tool kit for X is also under development.[24] The advantage of a tool kit is that it significantly reduces the effort required to create higher quality user interfaces, and it helps ensure consistency among the user interfaces of different application programs on the same machine. In addition, some window managers also provide a tool to help organize and use the contents of the tool kit. Examples of this are the Apple MacApp program[25] and Apollo's Open Dialog.[26] These tools are often called *user interface management systems*[15] and are necessary because programmers often find that tool kits are large and difficult to use.

## Window managers surveyed

Mentioning every window manager is impossible, since new ones appear all the time, and many are not documented in generally available publications. The selection here is not meant to be an indication of which window managers are best. The ones included are the ones that exemplify important variations. In addition, window managers are continually changing, so the descriptions for some window managers may no longer be accurate. The primary objective of this article is to illustrate the choices available in window manager user interfaces rather than to describe fully any particular window manager.

Some window managers allow their user interfaces to be changed. For example, X allows significant changes. For this class of window manager, the article describes one of the available user interfaces, and some of the variability is mentioned where appropriate. For X, the "uwm" window manager[27] for the IBM-RT computer is described and will be called "X/uwm."

Although this article discusses advantages and disadvantages of various user interface choices, this is not meant as a criticism of any window manager. As with all user interface decisions, there are often external considerations that influence the choice. The descriptions are meant to illustrate concepts rather than evaluate window managers.

The Table shows all the window managers mentioned in this article. The entries are approximately in chronological order.

## User interface of application programs

One interesting consideration is the extent to which the window manager's user interface affects the user interface of application programs that run under it. Even window managers that try to minimize their user interface will at least need to allow the user to change the listener and the positions of windows, and even this user interface will affect how an application program can interact with the user. Some window managers attempt to minimize their restrictions of the application's user interface so they can allow applications maximum flexibility.

Other window managers attempt to specify the user interface of applications to a large extent to ensure consistency. In any case, the choice of the user interface of the window manager *must* affect the user interfaces of the application programs. Even such window managers as X, whose own user interface can be changed, are not free of this problem. Different applications that run at the same time cannot all impose their choices on the same window manager, since the window manager user interface is global to all applications. Figure 4 shows a few sample window managers and how much they influence the user interfaces of applications.

## Presentation

Now the taxonomy of the user interface part of window managers will be presented. This section discusses the presentation aspects of this taxonomy, and a later section discusses the operations. Figure 5 shows the taxonomy of the presentation aspects of window managers. The following sections explain the various options shown in this figure.

### Tiled versus overlapped

The first major decision is whether windows are allowed to overlap or not. Some window systems (including Cedar and the original versions of Microsoft Windows[28]) require that windows be side by side and not overlap. As discussed earlier, this is called "tiling." The alternative is to allow windows to overlap, and this is provided by Smalltalk,[7,29,30] X/uwm, and many others.

Implementation and human factors issues guide the choice between tiling and overlapping. In tiling systems, the computer is typically in charge of managing the window placement and size, limiting the user's freedom. In covered window managers, the user must usually manage the windows. Putting the mouse in an arbitrary window is also easier with tiling, since all of the windows

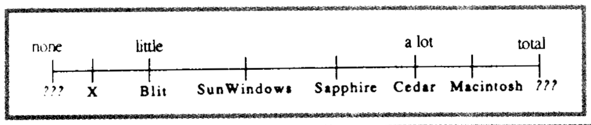


**Figure 4. The amount that the user interface of some sample window managers affects the user interface of application programs.**

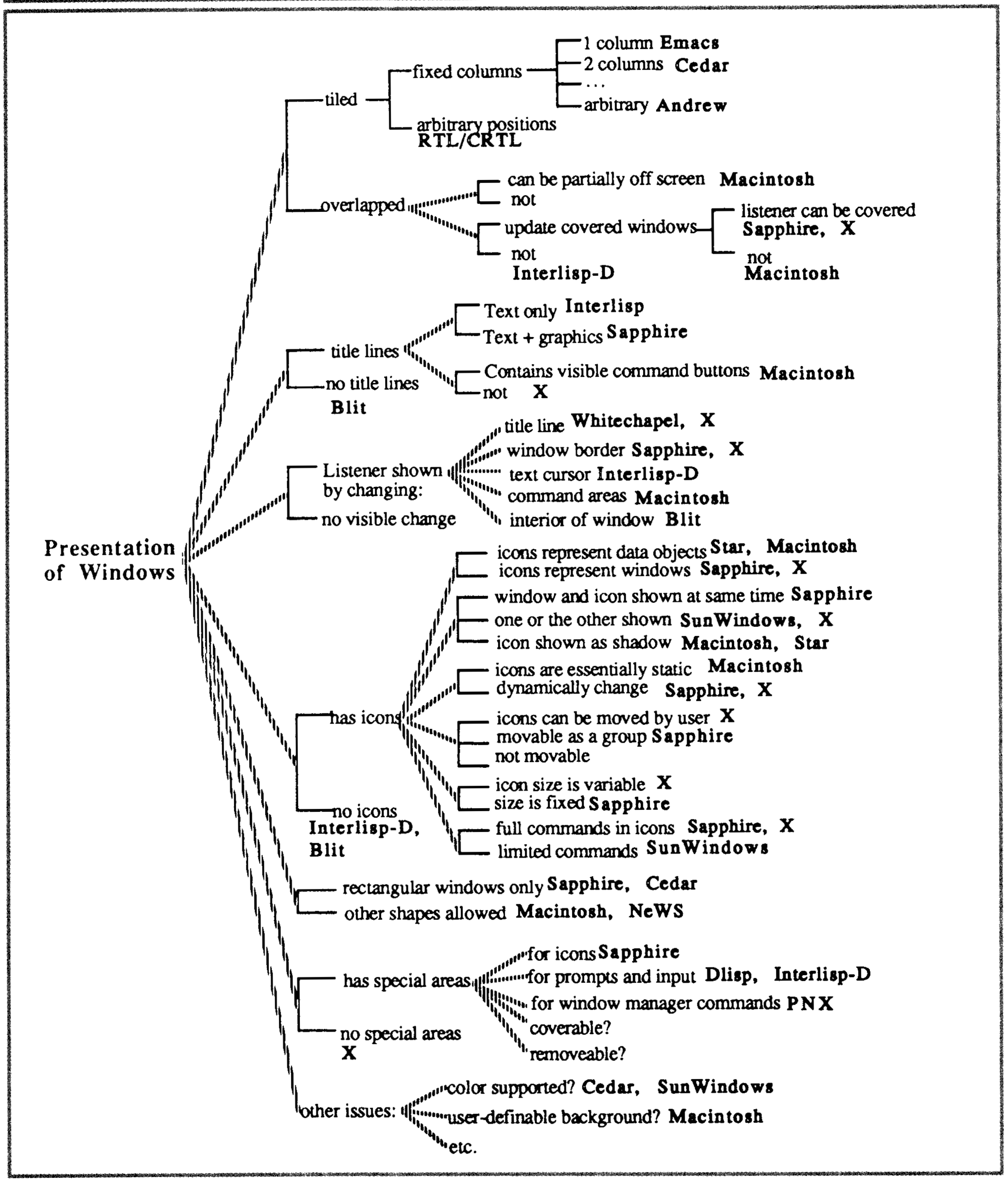

Presentation of Windows
tiled
fixed columns
1 column Emacs
2 columns Cedar
...
arbitrary Andrew
arbitrary positions RTL/CRTL
overlapped
can be partially off screen Macintosh
not
update covered windows
listener can be covered Sapphire, X
not Macintosh
not Interlisp-D
title lines
no title lines Blit
Text only Interlisp
Text + graphics Sapphire
Contains visible command buttons Macintosh
not X
Listener shown by changing:
no visible change
title line Whitechapel, X
window border Sapphire, X
text cursor Interlisp-D
command areas Macintosh
interior of window Blit
has icons
no icons Interlisp-D, Blit
icons represent data objects Star, Macintosh
icons represent windows Sapphire, X
window and icon shown at same time Sapphire
one or the other shown SunWindows, X
icon shown as shadow Macintosh, Star
icons are essentially static Macintosh
dynamically change Sapphire, X
icons can be moved by user X
movable as a group Sapphire
not movable
icon size is variable X
size is fixed Sapphire
full commands in icons Sapphire, X
limited commands SunWindows
rectangular windows only Sapphire, Cedar
other shapes allowed Macintosh, NeWS
has special areas
no special areas X
for icons Sapphire
for prompts and input Dlisp, Interlisp-D
for window manager commands PNX
coverable?
removeable?
other issues:
color supported? Cedar, SunWindows
user-definable background? Macintosh
etc.

**Figure 5. Taxonomy of the presentation aspects of the user interface of window managers. Solid lines are choices (exactly one of the options is chosen). A window manager can include any or all of the options at the ends of the diagonal gray lines. Example systems are shown in an outline font. There will typically be many other systems that also share the same features. The options shown are discussed in the article.**

are always visible. Often, the "best" style is a matter of personal taste, but one study discovered that, while users claimed to prefer covered window managers, they spent less time doing window management operations with tiled window managers.[31] The overall timing results for task completion were somewhat inconclusive, however.

The computer's screen size will also affect whether tiling or covered is preferred. With small screens, such as on the standard Macintosh, there is not enough room to use the tiling style. For the window-manager implementer, graphical output primitives are more difficult to provide with covered systems, since the output must be clipped to differently shaped regions. On the other hand, with tiled systems, the implementer must provide some sort of automatic screen layout facilities. Some window managers, such as Viewpoint[32] (which is the successor to Star), allow the user the choice of covered or tiled windows. Another possibility is for a window manager that uses covered windows to provide automatic layout. This is harder than with tiled windows, however, because it is less clear where windows should be placed.

When windows are tiled, the next decision is whether the windows must be arranged in fixed columns or whether the windows can be in arbitrary places on the screen (see Figure 6). A good discussion of the various options for tiled window systems can be found in Cohen's article.[33] Originally, the Andrew window manager used a constraint system to allow windows to be nonoverlapping and anywhere on the screen. The system would adjust the sizes of windows based on the constraints whenever a new window was created or an old window destroyed. Unfortunately, users did not like this for a number of reasons: It took too long for the windows to finish adjusting themselves after a change, windows all over the screen would change size when a new window was added or removed, and the screen layout resulting from one window changing was unpredictable. Therefore, Andrew now supports a much simpler approach instead, with a user-defined number of columns.[6]

The RTL/CRTL window manager from Siemens also uses constraints.[33] The current version runs on top of X11, and it reportedly does not have the problems discussed above for Andrew.

If there are fixed columns, then there can be either a specified number or an arbitrary number. Probably the first use of the window concept was in such full-screen text editors as Emacs,[34] which allowed multiple files to be edited at the same time by dividing the terminal screen into horizontal sections. This idea has been extended in window managers that allow multiple columns. For example, Cedar provides for exactly two columns on a black-and-white display (see Figure 2) along with one additional column on the optional color display.

If windows are allowed to overlap, then there are a number of secondary options. First is whether to allow windows to extend partially off screen (so that only part of the window is visible on the screen). Most covered window managers support this. Another option is whether to allow windows to be updated while they are covered. It is clearly more difficult to clip the output operations correctly so that covered windows can be updated, so such older window managers as Smalltalk and Interlisp-D[35] require that windows come to the top before being written to. Most modern window managers, however, allow output to windows while they are covered. If output can occur in portions that are covered, the next question is whether the listener window is allowed to be covered. There is no additional implementation difficulty in allowing this, but some window managers, such as the Macintosh, always bring the listener window to the top, to help users keep track of where they are typing. Other window managers, such as Sapphire and X/uwm, allow the listener to be covered so that users can have maximum flexibility in arranging their working environment.

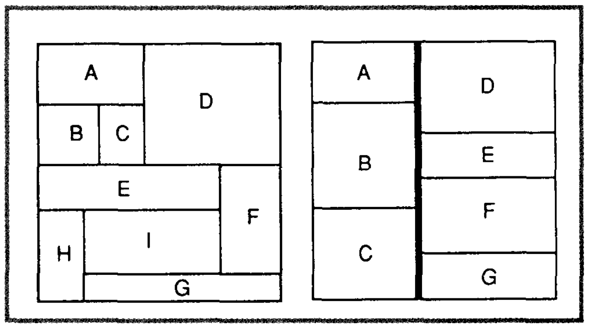


**Figure 6. Tiled windows can be in arbitrary places (left) or in fixed columns (right).**

## Title lines and borders

All window managers provide some "decoration" for the windows. This usually includes special "title lines" at the top and special borders around the entire window. The title line typically shows such global information as the current directory, the name of a file being edited, or the name of the program being run (see Figures 1 and 2). The Blit[36] window manager is one of the few window managers that does not use title lines. In addition, the title lines and borders may contain other decorations and command buttons.

## Showing the listener

An important presentation issue is how the listener is shown. Since there are multiple windows and only one listener, it is important that the user know which window is the listener. One method for specifying the listener is simply to move the mouse into a window. In this

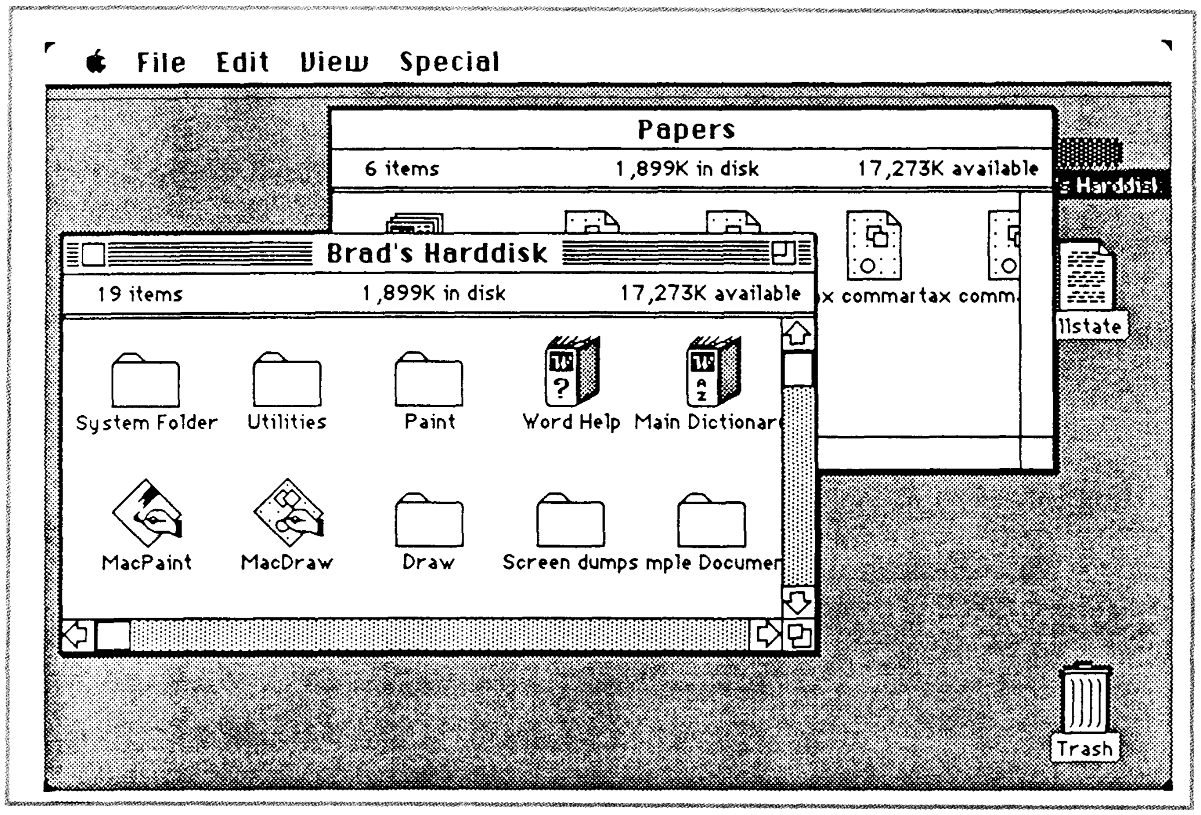


Figure 7. The listener on the Macintosh[3] is shown by drawing lines in the title line and displaying the command areas. The scroll bars and arrows on the right and bottom of the window move the display inside the window, the icon at the bottom right of the window is used to change the window's size, the icon at the top right is used to make the window full screen, and pressing in the square at the top left closes the window.

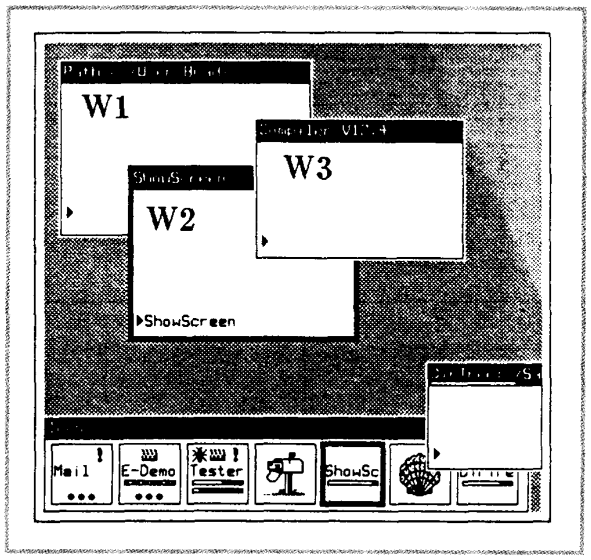


Figure 8. Sapphire[13] displays the listener window by making its border thicker. The window marked W2 is the listener. The icons are in the window at the bottom, and they can display status information about the window and the process running in it.

case the window containing the mouse tracking symbol is clearly the listener. The window manager might use some combination of other ways to show the listener, including changing the title line decorations (Whitechapel[23]), changing the border decorations (Sapphire[13]), changing the title and border (X/uwm[27]), changing the shape of the text input cursor or starting it blinking (Interlisp-D), removing the command areas (Macintosh), or even filling the window with a particular pattern when it is not the listener (dots are used in the Blit). Figure 7 shows a combination of changing the title line and command areas, and Figure 8 shows an example of using the border only.

### Icons

A major issue of presentation in modern systems is whether the window manager supports icons or not. These were invented by David Smith[37] and first used in a window manager in the Xerox Tajo environment[9] (which was later renamed XDE[38]). In Tajo the icons were originally just the title lines of the windows. A similar approach is used by the Andrew system, which leaves the title line where it was in the column, and simply hides the window contents.

Pictorial icons were first used in the Xerox Star,[1,2] where they give the user the illusion of operating in a physical environment. Each icon represents an object in an office environment (documents, folders, file cabinets, printers, etc.), so the user can learn how to operate these objects by analogy to the way operations are performed in the physical world. For example, to delete a file, the user moves the icon for the file to the icon that looks like a trash can (see Figure 3). Icons in the Star, as well as in the Apple Macintosh, therefore represent data objects and processes.

Most other window systems use icons in a different way. They represent windows. In these systems, the icon is simply another representation for a window, without the additional semantics of being data objects that can be operated on.

Most systems provide icons as an alternative representation for windows, so that a window is either full size or "shrunk" to an icon. Some of these systems, such as the Macintosh and Star, leave a "shadow" of the icon to show where it came from, whereas other systems (SunWindows and X/uwm) do not.

The Sapphire window manager uses icons in an entirely different way: The icon for a window is always fully visible, even if the window is on the screen. The icons in Sapphire are used to give commands to the windows and to show eight pieces of status information about the window and the process running in the window.[13] This includes percent-done progress indicators[39] to show how the job is doing and pictures to show when an error has occurred or when the process in the window is waiting for input (see Figure 8).

The Sapphire icons are an example of how the icons can dynamically change. Another example is the X/uwm window manager, where an icon can be a tiny version of the actual window, as shown in Figure 3.

In most other window managers, however, the icons are essentially static. Often, there are situations where the icon changes a little, however. For example, in the Macintosh, most icons are static, except that the trash can icon bulges when something is put into it. Similarly, on the Star, the mail icon changes to show when mail has arrived.

Other issues with icons are whether the user can move them around, whether their shape and size are fixed, and whether the user can give commands to the window using the icons. In the extreme, if icons are just an alternative representation for windows, the user should be able to type to the icon and have the text go to the application running in the window. This is allowed in some window managers, such as Sapphire, but others limit the commands available when the window is shown as an icon.

### Window shape

A minor point is what window shapes are supported. Usually, only rectangular windows are provided, but the NeWS window manager does support arbitrarily shaped windows (see Figure 9). Most of the windows in the Macintosh are rectangular (Figure 7), but they are allowed to be arbitrary shapes. Obviously, a tiling window manager must use shapes that can be tiled, such as rectangles or hexagons (all existing ones use rectangles). Many window managers simulate arbitrary shapes for icons, even if the windows must be rectangular.

### Special areas

Another presentation aspect of window-manager user interfaces is whether any screen areas are reserved for special functions. For example, DLisp[40] has a reserved area at the top and Interlisp-D has a reserved window for error messages and user prompts. Other examples are the Macintosh, which reserves the top line for a command menu, and PNX,[41] which has a special window-manager window for giving some window-manager commands. Other window managers have special areas for icons, which may be in the background, or in a special window (as in Sapphire). Icons may be in arbitrary places, as in X/uwm, on a regular grid, as in the Star, or in arbitrary places with a user-callable "neatening" onto a grid as in the Macintosh. Another issue is whether the icons can be covered by other windows (usually true if there are overlapped windows, false if tiled windows).

### Other issues

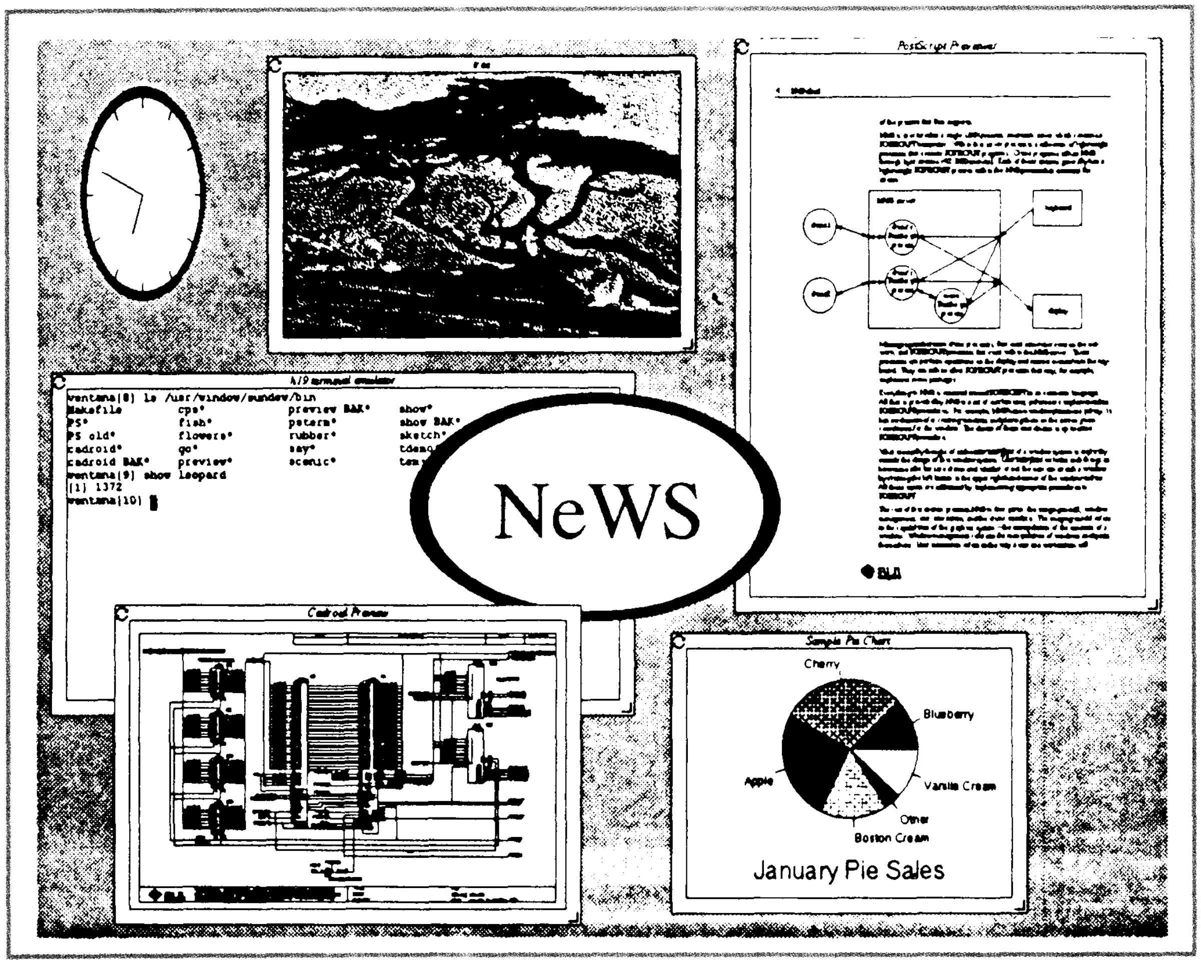
There are a number of other issues of presentation, such as whether color is supported, whether the user can manipulate the presentation (for example, changing the background pattern in the Macintosh), and to what extent application programs can change the presentation (for example, changing the title line's appearance).

## Operations

The second major component of the user interface of window managers is the set of operations that the user can execute to control the windows. This component can be further subdivided into the functionality of the operations provided, and the user interface of those operations. Taxonomies for these aspects are shown in Figures 10 and 11, respectively.

### Functionality

The first issue, of course, is which operations are supported. All window managers must supply some way for



```
ventana[8] ls /usr/window/sundev/bin
Makefile         cps*        preview BAK*   show*
PS*              fish*       psterm*        show BAK*
PS old*          flowers*    rubber*        sketch*
cadroid*         go*         say*           tdemo
cadroid BAK*     preview*    scenic*        tem
ventana[9] show leopard
[1] 1372
ventana[10]
```

**Figure 9. NeWS[12] supports nonrectangular windows.**

the user to change the listener. There usually are also commands to create new windows and delete old windows. A system with covered windows will also need a command to make the window become uncovered (top), and there is usually a command to make a window be covered by all other windows (bottom). The bottom command is often used to remove windows that are hiding a fully covered window.

Most window managers allow the user to move windows and change their size, but these operations may be restricted in a tiled window manager. For example, in Cedar, you can only move a window to the other column or change its size in the current column, and the width of all windows in a column must be adjusted together. Other tiled systems allow a window to be placed in the "seam" between any two other windows. With a covered window system, an issue is whether the windows can change size or move while they are partially covered (true in Sapphire, false in X/uwm). On the Macintosh it is possible to move a covered window by holding down the command key while pressing in the window's title line.

Another issue is whether a window can be modified only from a particular corner (as in the grow operation for the Macintosh, which works only from the lower right corner), from a set of control points (Sapphire allows windows to be moved and changed in size from points on each side and corner), or from anywhere in the window (as in X/uwm).

For window managers that have icons, there are often operations to shrink the window down to the icon and to expand from the icon to normal size. There might also be operations to move the icon itself (no system I am aware of allows the user to change the icon's size). Another operation that is often provided is the ability to make a window full screen or full size. The old size and position are remembered so a single operation can make the window go back. This command is available in Sapphire and for some windows on the Macintosh.

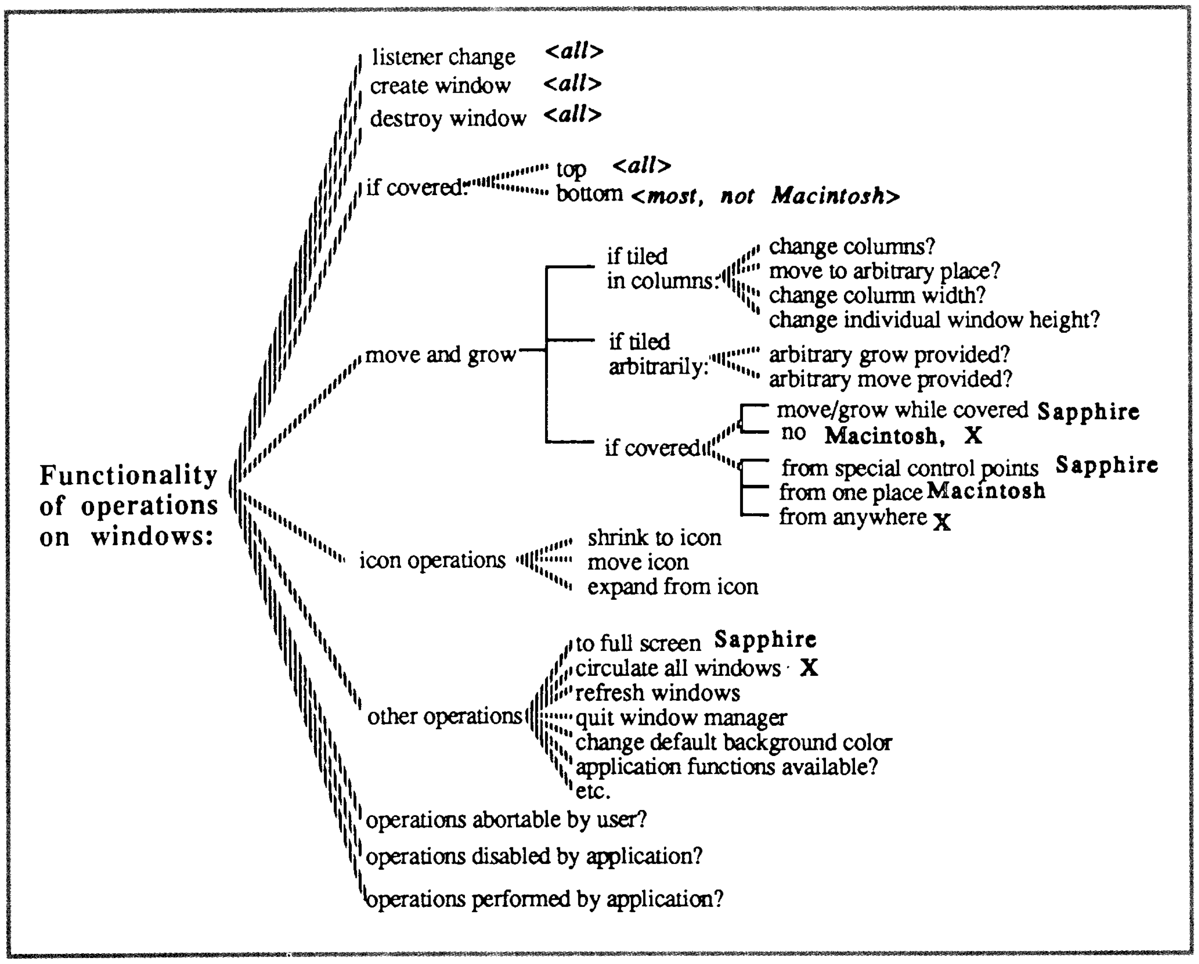


**Figure 10. Taxonomy for the functionality of window manager operations.**

Some window managers provide additional operations. For example, most window managers allow a change from black text on white to white on black; X/uwm allows the most covered window to be circulated to the top; and Sapphire and X/uwm have commands to refresh the contents of windows.

For systems where the window manager itself is optional, such as SunWindows and X/uwm, there is often a command to quit the entire window manager. Some window managers allow applications to insert their own commands into the normal window manager command mechanism (for example, into the Interlisp-D menus). There may also be an Undo command to reverse the previous window manager command.

Another important question is whether the user can abort a command after it has started. For example, in the Macintosh, if you begin to move a window and change your mind, you can move the cursor into the command area and the move operation will be aborted. On the other hand, there is no way to stop the grow operation on the Macintosh. In Sapphire, any operation can be aborted by hitting the Delete key on the keyboard.

Another issue is whether application programs can affect the functionality of the commands. For example, a program implementing a terminal emulator might want to restrict its window to be exactly 24 by 80 characters big and therefore disable the window-size-change command. In a different situation an application may be designed for novices and therefore want to disable any "dangerous" window manager commands. Another example is how dialogue boxes refuse to allow the user to change the listener on the Macintosh.

A related question is whether applications can execute the commands. For example, can applications move existing windows or change their size? Can an application change the listener? The latter functionality is useful to provide alarms for special asynchronous situations. For example, the PrintMonitor on the Macin-

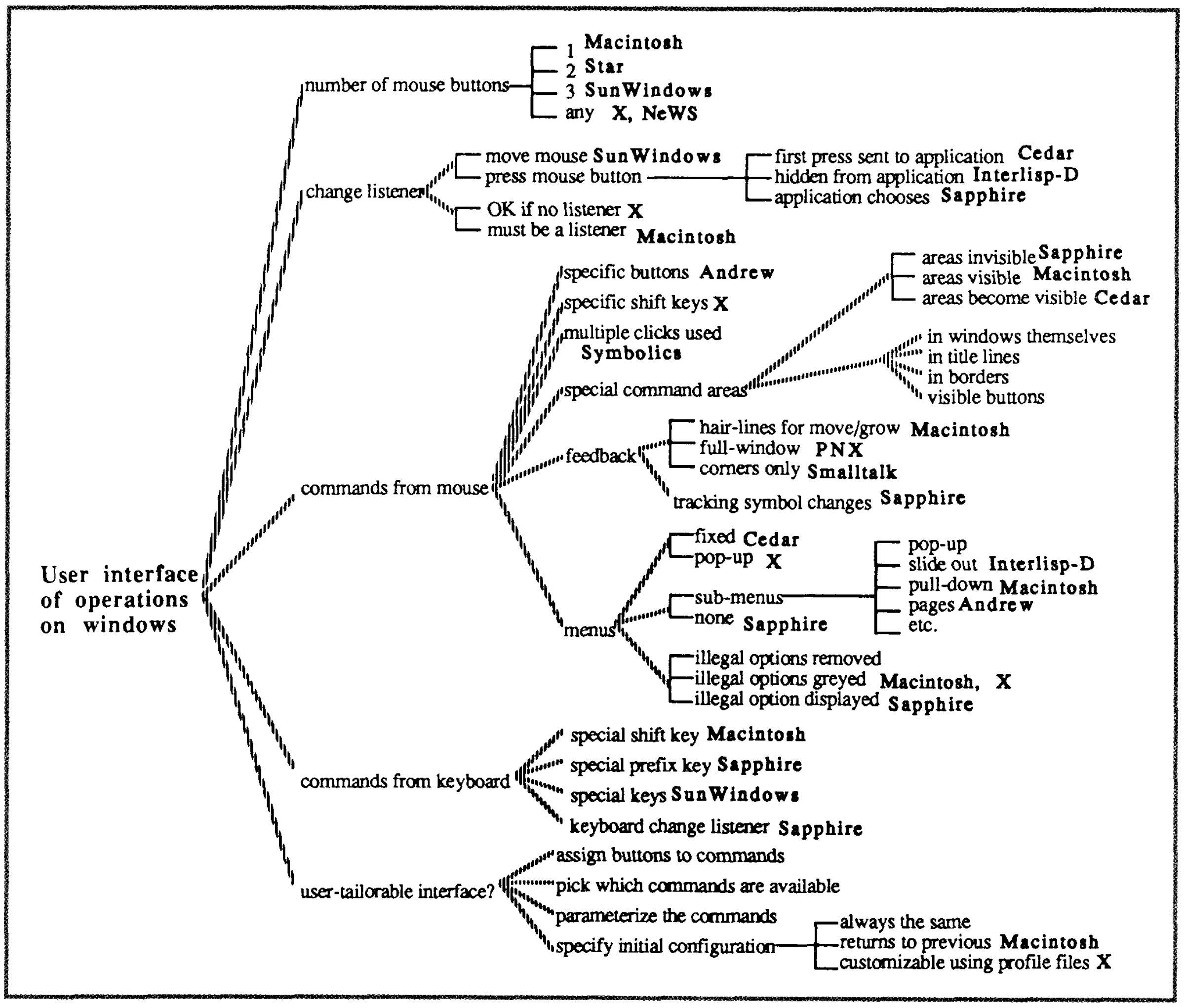

User interface of operations on windows
number of mouse buttons
1 Macintosh
2 Star
3 SunWindows
any X, NeWS
change listener
move mouse SunWindows
press mouse button
first press sent to application Cedar
hidden from application Interlisp-D
application chooses Sapphire
OK if no listener X
must be a listener Macintosh
commands from mouse
specific buttons Andrew
specific shift keys X
multiple clicks used Symbolics
special command areas
areas invisible Sapphire
areas visible Macintosh
areas become visible Cedar
in windows themselves
in title lines
in borders
visible buttons
feedback
hair-lines for move/grow Macintosh
full-window PNX
corners only Smalltalk
tracking symbol changes Sapphire
menus
fixed Cedar
pop-up X
sub-menus
none Sapphire
pop-up
slide out Interlisp-D
pull-down Macintosh
pages Andrew
etc.
illegal options removed
illegal options greyed Macintosh, X
illegal option displayed Sapphire
commands from keyboard
special shift key Macintosh
special prefix key Sapphire
special keys SunWindows
keyboard change listener Sapphire
user-tailorable interface?
assign buttons to commands
pick which commands are available
parameterize the commands
specify initial configuration
always the same
returns to previous Macintosh
customizable using profile files X

**Figure 11. Taxonomy for the user interface of window manager operations.**

tosh pops up a window and makes it the listener when there is a problem with the printer.

The advantage of a large number of commands is that the user can perform operations in a variety of ways. The disadvantage, however, is that it may be more difficult for the user to know which command to use, and therefore make the window manager more difficult for novices. In addition, if a command has lots of variants and options, it may take longer to invoke any single command, since the various options must be specified. As an example, allowing a window to be grown from any side or corner usually involves three steps: first giving the grow command, second specifying the place to grow from, and finally specifying the new size. When there is less flexibility, as in the Macintosh, the second step is eliminated.

Future window managers will probably need to provide additional functionality in their user interfaces to help the user control the windows. Whereas icons and automatic layout for windows are good first steps, users still find that they frequently lose windows and have difficulty reconstructing their working environment. This is especially important since studies have shown that people do not work on a particular task for more than 15 minutes on average before they switch to a different task, presumably in a different window.[42] Some research window managers provide groupings of windows, so that the user can set up a related group of tasks running in windows in a particular layout, and go directly from one group to another. Cedar contains a fixed overview of 16 screens, each of which has a set of windows, and the Rooms system[43,44] provides arbitrary groupings of windows.

## User interface of operations

The next important issue with operations is how they can be specified by the user. Many window managers provide multiple ways of giving the same command. For example, there may be menus that contain all commands for novices, as well as *accelerators*—faster ways for expert users to give the most frequent commands using the mouse and keyboard. Clearly, the considerations about what user interface to use for the window manager must be influenced by general user interface principles, which are more fully described in other places, for example by Foley and van Dam.[45]

### Number of mouse buttons

One of the most obvious differences between window managers is how many buttons on the mouse they are designed to use. The Macintosh mouse has only one button because it is intended to be very easy to use for novices. With more buttons, the novice might forget which button performs which operation and be nervous about pressing them. The designers of the Star system did extensive testing and decided two buttons were easiest to learn and most efficient.[46] Many other window managers are designed for three buttons (SunWindows and Sapphire, for example). Some window managers simulate a middle button on a two-button mouse by holding down both buttons at the same time. Naturally, the customizable window managers, such as X and NeWS, can support any number of buttons on the mouse.

### Changing the listener

Of particular interest is how the user changes the listener. Some systems, including SunWindows and Smalltalk, change the listener to whatever window contains the cursor. Other window managers—including Interlisp-D, Star, Sapphire, Blit, Cedar, and Macintosh—require an explicit press to change the listener. Some window managers, such as X and NeWS, allow the user to pick which technique is used.

If an explicit press is required, then an issue is whether this press should be sent through to be used by the application program. If so, as in Cedar, changing the listener must always do something to the application (such as changing the insert point in an editor). If the button press is not sent through, as in Interlisp-D, then the user might be confused that the operation did not happen. Some systems, such as Sapphire, let the application decide whether to interpret the first press, but this means that applications may operate inconsistently.

The advantages of changing the listener on cursor movement are that this operation is easier and faster, the mouse cursor provides a nice form of feedback for which window is the listener, and the problem of whether to send the press through to applications is avoided. The advantages of using a press to change listeners are that the listener does not change accidentally if the mouse is bumped, input devices like a stylus that do not retain their position can easily be used, the position of the mouse can still be used even when it is outside of the listener window, the mouse can be moved outside the window so no part of the window is hidden under the cursor, and if the input device is a tablet, it can be used in either absolute (digitizing) or relative (mouselike) tracking modes.[13]

Another issue is whether the window manager allows there to be *no* listener. In X/uwm, for example, when the cursor is outside of all windows (that is, over the background), keyboard typing just disappears. In Sapphire, keys typed when there is no listener are saved and given to the next window that becomes the listener. On the other hand, the Macintosh will not allow no listener; if you press over the background, nothing happens, and if the listener window is deleted, then the next topmost window is made the listener. This tends to be much more intuitive and keyboard typing is never lost.

A question for future window managers will be how to handle additional input devices. Currently, window managers typically support only the keyboard and one pointing device. Research has demonstrated, however, that using multiple input devices, such as touch tablets, knobs, and switches, along with the pointing device (for example, one in each hand) can increase the user's effectiveness.[47] In addition, speech recognition is slowly becoming practical. All of these input techniques will have to be switched among windows as the keyboard is now, and the issue will be whether to switch all devices at the same time, to have different techniques for switching each device separately, or to allow certain devices to be grabbed permanently by a particular window.

### Other commands from a mouse

There are various ways that window managers allow commands to be given using the mouse. Some systems, such as Andrew, reserve a special mouse button to give window manager commands. Other systems, such as X/uwm, reserve a special shift key on the keyboard. For example, to give a window manager command on the IBM RT computer, you need to hold down the "Alt" keyboard key while pressing the mouse buttons. Still other systems, such as Symbolics[48] and Macintosh, use multiple clicking.

Pressing the mouse buttons while the cursor is in different areas typically has different meanings. Most window managers, for example, provide special commands when the mouse is pressed in the title lines of windows. Some window managers also allow the user to press in the borders of windows, which might be used for changing a window's size or position. The Macintosh window manager, among others, has special "buttons" in the window title line and border which are used for closing a window or changing its size (see Figure 7). On the other hand, some window managers allow the commands to be given if the cursor is anywhere inside the

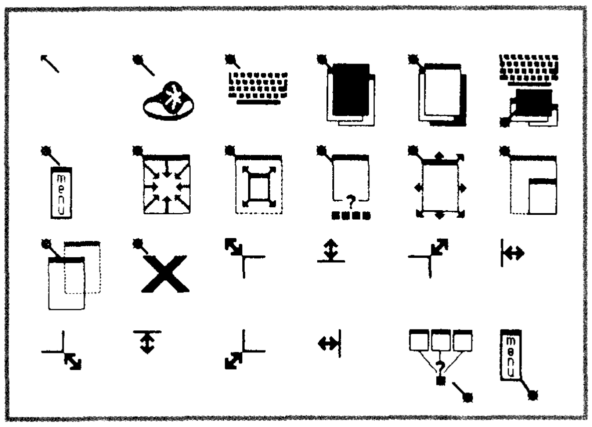


**Figure 12. Various cursor pictures used in Sapphire[13] to show which operation is in progress.**

window. Sapphire and the Macintosh, for example, cause a window to become the listener when the cursor is pressed anywhere inside it.

The special areas for giving commands are sometimes not visible in the window. For example, Tajo and Sapphire divide the title line into three regions horizontally and assign different functions to each region. The divisions are not shown, but the user is assumed to be able to differentiate the middle from the left and right of the title line for any window that has a reasonable size. Other window managers only use visible areas. The trade-off is obviously using screen area for command labels so they will be easier to find and more obvious for novices, versus using the screen space to display other useful information. Cedar has a unique solution to this problem: Some of the command buttons are hidden underneath the title line, and the title line is replaced with the command menu whenever the cursor goes into the title line.

Another issue when giving commands with the mouse is what kind of feedback the user sees. Most systems provide hairlines the size of the window when a window is moved. The PNX window manager actually has the entire window follow the mouse in real time, and originally Smalltalk showed only one corner.

Often, there will be feedback in the cursor picture as to the operation being performed. For example, X/uwm shows a picture of the button that is down. Sapphire uses the cursor picture to show which operation may happen. When the button is pressed in a particular area, the cursor changes to show what will happen (see Figure 12). If the button is released, the operation happens, but if the cursor is moved away before releasing, the operation is aborted. Different cursor pictures are also often used to show what mode the user interface is in.

### *Menus*

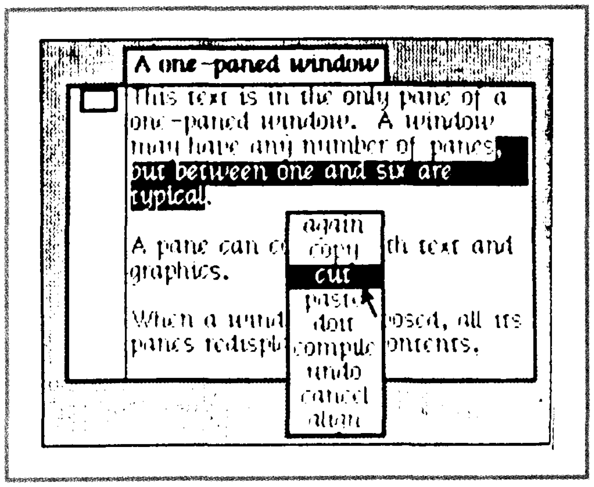
Commands in window managers are usually given using menus. A menu is a list of options, and in a window manager, the desired option is usually picked by pointing at it with the mouse. Menus were classified as an add-on earlier, since they are usually supplied to application programs as a service. On the other hand, most window managers also use menus as part of their own user interface, so it is necessary to discuss them here. This discussion is not meant to be comprehensive, however, since there are many different ways to present menus and the options listed here are only those that have been commonly used with window managers.

Menus can be always visible (usually at a fixed place on the screen or a fixed place in each window), in which case they are called *fixed* menus. Alternatively they can appear when the user performs some action, and then are called *pop-up* menus (see Figure 13). Pop-up menus are often used because they do not take up screen area when not in use, and they appear at the cursor so less hand movement is needed. On the other hand, they have the disadvantages that novices may not know how to make them appear, and they usually take longer to use since the user must first perform the action to make them appear, then search through the menu for the correct item, and then make the selection.

Fixed menus also allow the user to *mouse-ahead*. Mouse-ahead is giving commands with the mouse before the system is ready to accept them. This is analogous to type-ahead from the keyboard. Pop-up menus are hard to use with mouse-ahead because the menu is not displayed, so the user cannot see where the desired item is. Fixed menus are used by the Macintosh on the top line of the screen (see Figure 14b), in DLisp in windows that can appear anywhere on the screen (see Figure 13b), and in Cedar as the menus in the title lines (see Figure 2). Fixed menus in a special place on the screen, as in the Macintosh, are probably appropriate only when the screen is small, so that the maximum hand movement to move the cursor to them is not too large.

Many window managers use pop-up menus to give window-manager commands. For example, in X/uwm when the Alt key and the left mouse button are held down, a menu pops up which contains the standard window-manager commands. The middle button with the Alt key pops up a different set of commands. Sapphire provides a menu of commands when the right mouse button is pressed over the left or right title line areas. A distinction among pop-up menus is whether they appear on a down press and the selection is made on the release (X/uwm), or whether a second down press is needed to make the selection (Symbolics and Sapphire). Some menu systems support both styles (RTL/CRTL[33]).

Another issue with menus is whether submenus are supported. A submenu is a menu that appears after a

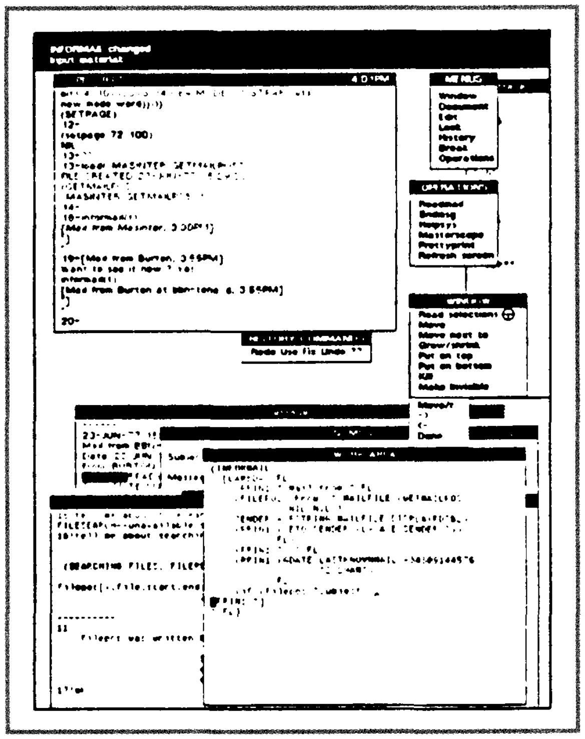

A one-paned window

This text is in the only pane of a one-paned window. A window may have any number of panes, but between one and six are typical.

A pane can c… th text and graphics.

When a wind… osed, all its panes redispl… ontents.

a



b

**Figure 13. A pop-up menu (a) from smalltalk, and fixed menus (b) in DLisp.**

particular item in a menu is selected. This will typically provide more options or parameters to that particular operation.

There are various ways to provide submenus. The most obvious way is simply for them to pop up after the main selection has been made. Interlisp-D and SunWindows have the submenus appear when the cursor is slid off a menu item to the right (see Figure 14a). Macintosh uses a similar idea, called *pull-down* menus, but here the main menu is horizontal and the submenus appear when the cursor is over any top-level item (see Figure 14b). Andrew has the menus stacked like pages of a book (see Figure 14c), so the user can flip through them quickly. Many other options are possible.

A final feature of menus in window managers is the handling of items that are currently illegal. For example, a particular application may prevent its window's size from being changed. The illegal item might be left out of the menu altogether, it might be included but shown in gray, as in X/uwm and the Macintosh (see Figure 14b), or it might be shown normally and just fail to operate, as in Sapphire.

The advantage of showing the option in gray is that the illegal items are obvious, but the other items in the menu always appear in the same places so the user gets familiar with where items are placed. The advantage of showing only the legal options is that more items can be included if they all appear in independent sets.

### *Commands from the keyboard*

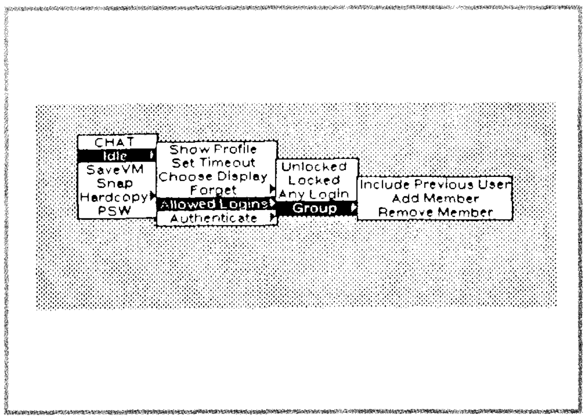
Some window managers allow operations to be executed using the keyboard. This might be done by using a reserved shift key (for example, the command key on the Macintosh); by using reserved keys for special functions (some function keys in Apollo,[49] for example, and SunWindows); or by reserving a prefix key which must be typed before window manager commands, as in Sapphire.

The keyboard commands are usually considered accelerators which experts will use when pop-up menus are too slow. Sometimes the keyboard commands are provided for people who do not like or cannot use the mouse. EasyAccess on the Macintosh, for example, was designed to allow handicapped people to replace all mouse commands with keyboard actions. The Apollo window manager is one of the few that was designed with the keyboard as its primary input device; the early Apollos did not have mice.[49]

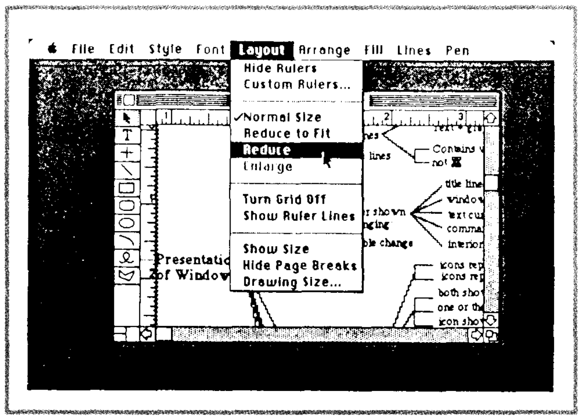
Other issues with keyboard commands are whether type-ahead is supported, whether the keyboard can change the listener (false in the Macintosh, true in Sapphire and SunWindows), and whether the keyboard can make selections in the menus.

### *Other user interface issues*

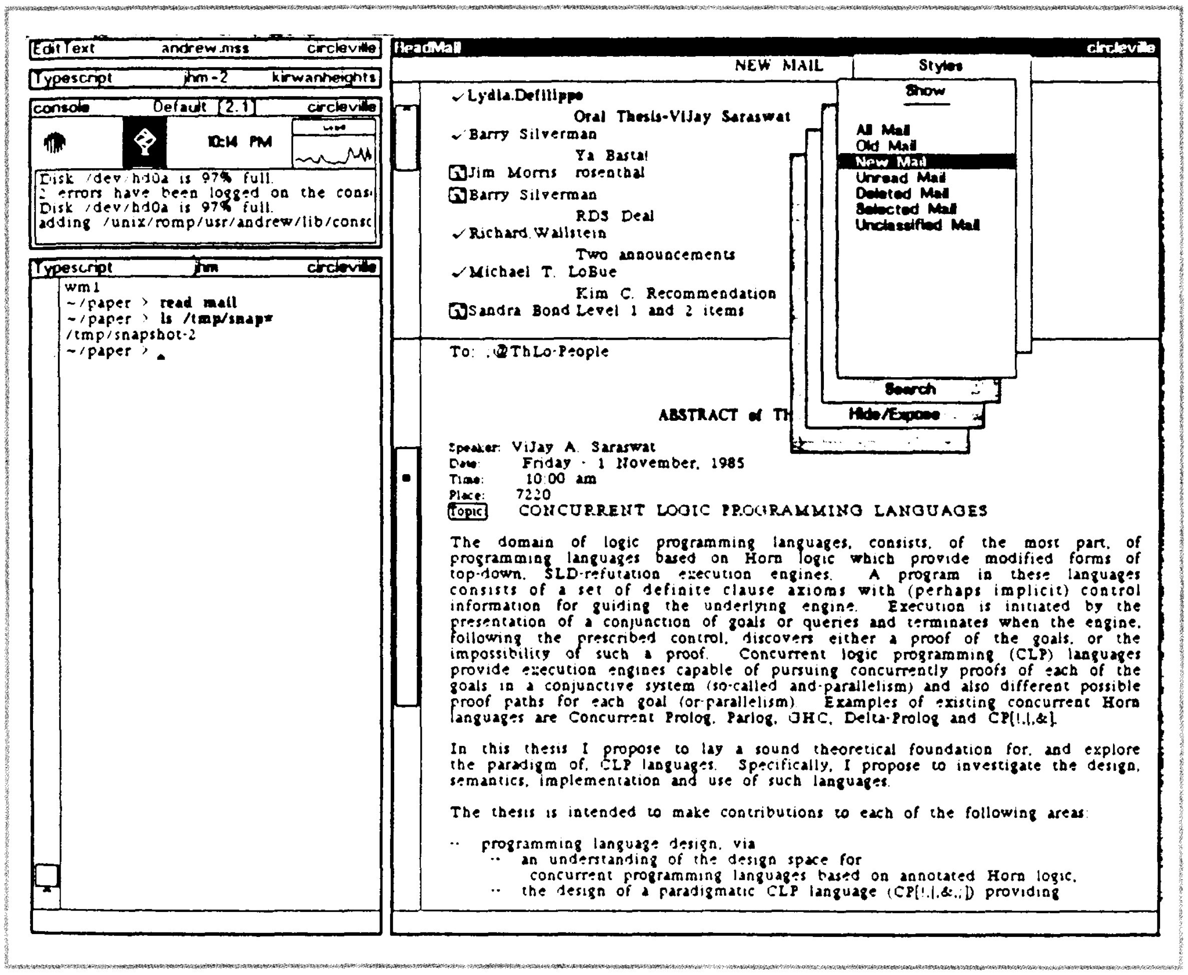
Some commands require arguments. For example, the user must specify a new size for a window when the change-size or create-window command is given. Some



a



b



```
Disk /dev/hd0a is 97% full.
2 errors have been logged on the cons
Disk /dev/hd0a is 97% full.
adding /unix/romp/usr/andrew/lib/consc
```

```
wm1
~/paper > read mail
~/paper > ls /tmp/snap*
/tmp/snapshot-2
~/paper >
```

✓Lydia.Defillippo
Oral Thesis-VIJay Saraswat
✓Barry Silverman
Ya Basta!
Jim Morris rosenthal
Barry Silverman
RDS Deal
✓Richard.Wallstein
Two announcements
✓Michael T. LoBue
Kim C. Recommendation
Sandra Bond Level 1 and 2 items

To: .@ThLo-People

ABSTRACT of Th

Speaker: VIJay A. Saraswat
Date: Friday - 1 November, 1985
Time: 10:00 am
Place: 7220
Topic CONCURRENT LOGIC PROGRAMMING LANGUAGES

The domain of logic programming languages, consists, of the most part, of programming languages based on Horn logic which provide modified forms of top-down, SLD-refutation execution engines. A program in these languages consists of a set of definite clause axioms with (perhaps implicit) control information for guiding the underlying engine. Execution is initiated by the presentation of a conjunction of goals or queries and terminates when the engine, following the prescribed control, discovers either a proof of the goals, or the impossibility of such a proof. Concurrent logic programming (CLP) languages provide execution engines capable of pursuing concurrently proofs of each of the goals in a conjunctive system (so-called and-parallelism) and also different possible proof paths for each goal (or-parallelism). Examples of existing concurrent Horn languages are Concurrent Prolog, Parlog, GHC, Delta-Prolog and CP[!,|,&].

In this thesis I propose to lay a sound theoretical foundation for, and explore the paradigm of, CLP languages. Specifically, I propose to investigate the design, semantics, implementation and use of such languages.

The thesis is intended to make contributions to each of the following areas:

-- programming language design, via
  -- an understanding of the design space for
     concurrent programming languages based on annotated Horn logic,
  -- the design of a paradigmatic CLP language (CP[!,|,&,;]) providing

c

Figure 14. Submenus may appear when the user slides the cursor out of the right of the item marked with an arrow (a) as in Interlisp-D,[35] when the cursor is over an item (b) as in the Macintosh,[3] or they may be stacked like pages (c) as in Andrew.[16] The Enlarge command in (b) is shown in gray because it is currently illegal.

window managers provide accelerators to allow special values of the parameters to be specified easily. For example, in X/uwm, when creating a window, a special button combination will make the window a default size that depends on the application (for example, 80-by-24-characters high in the default font for a terminal emulator). A different button combination allows the user to specify the size using the mouse.

Another important issue is to what extent the user can change the user interface of the window manager. Window systems such as X and NeWS were designed to allow the user total flexibility, so almost any aspect can be changed. In X/uwm, many of the changes can be made by editing specially formatted text files (for example, .uwmrc), but other changes require writing programs. Other window managers allow customization to a lesser extent. For example, Cedar allows the menu items and their functions to be assigned by the user employing "TIP" tables,[11] but the general window layout is fixed. A related issue is how much of the initial screen layout can be specified for when the system is powered on. Some systems always start the same way, others allow the user to define the initial configuration in profile files (X/uwm), and others return the screen to the last configuration (Macintosh).

## Conclusions

This article has attempted to list some of the important common aspects of the user interfaces of window managers. The various features can be broken down into fairly simple taxonomies, which will be useful when studying and comparing current window managers or designing new ones. Clearly, there will always be new innovations that are not in these taxonomies, but they are expected to cover the important parts of new designs, and may even help future designers see where new techniques might be used.

One obvious conclusion from looking at the taxonomies is that there is not a great deal of difference among different window managers. In fact, when faced with a new window manager, a user could probably deduce how to operate it by pressing the various mouse buttons with various keyboard shift keys (Shift, Control, Meta, Hyper, Super, Alt, etc.) held down over the special areas (in the window, the title line, the border, on any special decorations which might be buttons, etc.). This is also encouraging for efforts at standardizing window managers, since there is less variability that must be accommodated. ■

## Acknowledgments

First, I want to thank the Alvey workshop[5] for sponsoring the conference that inspired this article. Mark Weiser provided many helpful comments and edited an earlier version of this article. Warren Teitelman participated in some early discussions. A number of people at various companies also supplied useful information about their window managers. For help and support with this article, I would like to thank Bernita Myers, David Anderson, Doug Bunting, Richard Cohn, and Randy Pausch.

Work on this article was started while the author was in the Dynamic Graphics Project, Computer Systems Research Institute, University of Toronto. Funding at Carnegie Mellon was provided by the Defense Advanced Research Projects Agency (DoD), ARPA order no. 4976 under contract F33615-87-C-1499 and monitored by the Avionics Laboratory, Air Force Wright Aeronautical Laboratories, Aeronautical Systems Division (AFSC), Wright-Patterson AFB, OH 45433-6543.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency of the US government.

## References

1. D.C. Smith et al., "Designing the Star User Interface," *Byte*, Vol. 7, No. 4, Apr. 1982, pp. 242-282.
2. D.C. Smith et al., "The Star User Interface: An Overview," *Proc. Nat'l Computer Conf.*, AFIPS Press, Arlington, Va., 1982, pp. 515-528.
3. W. Gregg, "The Apple Macintosh Computer," *Byte*, Vol. 9, No. 2, Feb. 1984, pp. 30-54.
4. R.W. Scheifler and J. Gettys, "The X Window System," *ACM Trans. on Graphics*, Vol. 5, No. 2, Apr. 1986, pp. 79-109.
5. F.R.A. Hopgood et al., eds., *Methodology of Window Management* (Proc. Alvey Workshop), Springer-Verlag, Berlin, 1986, 250 pp.
6. G. Krasner, *Smalltalk-80: Bits of History, Words of Advice*, Addison-Wesley, Reading, Mass., 1983.
7. L. Tesler, "The Smalltalk Environment," *Byte*, Vol. 6, No. 8, Aug. 1981, pp. 90-147.
8. W. Teitelman, "Ten Years of Window Systems—A Retrospective View," *Methodology of Window Management* (Proc. Alvey Workshop), 1985, Springer-Verlag, Berlin, 1986, pp. 35-46.
9. D. Wallace, "Tajo Functional Specification, version 6.0," tech. report, Xerox, Oct.1980.
10. W. Teitelman, "A Tour Through CEDAR," *IEEE Software*, Vol.1, No. 2, Apr. 1984, pp. 44-73.
11. D. Swinehart et al., "A Structural View of the Cedar Programming Environment," *ACM Trans. Programming Languages and Systems*, Vol. 8, No. 4, Oct. 1986, pp. 419-490.
12. Sun Microsystems, Inc., *NeWS Preliminary Technical Overview*, Mountain View, Calif., 1986.
13. B.A. Myers, "The User Interface for Sapphire," *CG&A*, Vol. 4, No. 12, Dec. 1984, pp. 13-23.
14. 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.
15. B.A. Myers, "Tools for Creating User Interfaces: An Introduction and Survey," Tech. Report CMU-CS-88-107, Carnegie Mellon University, Computer Science Dept. Also to appear in *IEEE Software*, Jan. 1989.

16. J.H. Morris et al., "Andrew: A Distributed Personal Computing Environment," *CACM*, Vol. 29, No. 2, Mar. 1986, pp. 184-201.

17. Sun Microsystems, Inc. *SunWindows Programmers' Guide*, Mountain View Calif., 1984.

18. G. Williams, "The Lisa Computer System," *Byte*, Vol. 8, No. 2, Feb. 1983, pp. 33-50.

19. Apple Computer, *Inside Macintosh*, Addison-Wesley, Reading, Mass., 1985.

20. J. Warnock and D.K. Wyatt, "A Device-Independent Graphics Imaging Model for Use with Raster Devices" Computer Graphics, (Proc. SIGGRAPH), Vol. 16, No. 3, July 1982, pp. 313-319.

21 J. Gosling, "SunDew—A Distributed and Extensible Window System," *Methodology of Window Management*, Springer-Verlag, Berlin (Proc. Alvey Workshop, 1985), pp. 47-57. (A slightly different version of this paper with the same title also appeared in the Proc. 1986 Winter Usenix Tech. Conf., Jan. 1986, pp. 98-103.

22. Adobe Systems, *Postscript Language Reference Manual*, Addison-Wesley, Reading, Mass., 1985.

23. D. Sweetman, "A Modular Window System for Unix," *Methodology of Window Management* (Proc. Alvey Workshop, 1985), Springer-Verlag, Berlin, 1986, pp. 73-80.

24. MIT and DEC, *X Tool kit Library—C Language Interface; Tool kit Beta Version 0.1; X Protocol Version 11*, Boston, 1987.

25. K.J. Schmucker, "MacApp: An Application Framework," *Byte*, Vol. 22, No. 8, Aug. 1986, pp.189-193.

26. Apollo Computer, "Open Dialog Interface Management System Supports IBM, DEC, and Sun," Product Announcement, Chelmsford, Mass., 1987.

27. M. Gancarz, "Uwm: A User Interface for X Windows," *Conf. Proc. Summer Tech. Conf.*, Usenix, Denver, Jan. 1986, pp. 429-440.

28. V. Puglia et al., "Operating in a New Environment," *PC Magazine*, Feb. 1986, pp. 109-132.

29. A. Goldberg, *Smalltalk-80: The Interactive Programming Environment*, Addison-Wesley, Reading, Mass., 1984.

30. A. Goldberg and D. Robson, *Smalltalk-80 The Language and Its Implementation*, Addison-Wesley, Reading, Mass., 1983.

31. S.A. Bly and J.K. Rosenberg, "A Comparison of Tiled and Overlapping Windows," *Proc. SIGCHI, Human Factors in Computing Systems*, ACM, New York, 1986, pp. 101-106.

32. Xerox Office Systems Division, *ViewPoint Users Manual*, Palo Alto, Calif., 1985.

33. E.S. Cohen, E.T. Smith, and L.A. Iverson, "Constraint-Based Tiled Windows," *Proc. 1st Int'l Conf. on Computer Workstations*, CS Press, Los Alamitos, Calif., Nov. 1984, pp. 2-11.

34. R.M. Stallman, "Emacs: The Extensible, Customizable, Self-Documenting Display Editor," Tech. Report 519, MIT Artificial Intelligence Lab, Cambridge, Mass., 1979.

35. W. Teitelman and L. Masinter, "The Interlisp Programming Environment," *Computer*, Vol. 14, No. 4, Apr. 1981, pp. 25-33.

36. R. Pike, "Graphics in Overlapping Bitmap Layers," *ACM Trans. Graphics*, Vol. 2, No., 2, Apr. 1983, pp. 135-160. Also appears in *Computer Graphics* (Proc. SIGGRAPH), Vol. 17, No. 3, July 1983, pp. 331-355.

37. D.C. Smith, *Pygmalion: A Computer Program to Model and Stimulate Creative Thought*, Birkhauser, Basel, 1977.

38. Xerox Office Systems Division, *Xerox Development Environment: Concept and Principles*, Part #610E00130, Palo Alto, Calif., 1981.

39. B.A. Myers, "The Importance of Percent-Done Progress Indicators for Computer-Human Interfaces," (Proc. SIGCH)I, ACM, New York, 1985, pp. 11-17.

40. W. Teitelman, "A Display Oriented Programmer's Assistant," *Int'l J. Man-Machine Studies*, Vol. 11, 1979, pp. 157-187. Also Xerox PARC Tech. Report CSL-77-3, Palo Alto, Calif., Mar. 8, 1977.

41. Int'l Computers Ltd., *ICL PERQ Guide to PNX*, Int'l Computers Ltd., UK Software and Literature Supply Sector, Reading, RG3 1NR, UK, 1983.

42. L. Bannon et al., "Evaluating and Analysis of Users' Activity Organization," *Proc. SIGCHI, Human Factors in Computing Systems*, ACM, New York, 1983, pp. 54-57.

43. S.K. Card and A. Henderson, Jr., "A Multiple Virtual-Workspace Interface to Support User Task Switching," *Proc. SIGCHI+GI, Human Factors in Computing Systems, Graphics Interface*, ACM, New York, 1987, pp. 53-59.

44. D.A. Henderson, Jr., and S.K. Card, "Rooms: The Use of Multiple Virtual Workspace to Reduce Space Contention in a Window-Based Graphical User Interface," *ACM Trans Graphics*, Vol. 4., No. 3, July 1986, pp. 211-243.

45. J.D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading Mass., 1982.

46. W.L. Bewley et al., "Human Factors Testing in the Design of Xerox's 8010 'Star' Office Workstation," *Proc. SIGCHI, Human Factors in Computing Systems*, ACM, New York, Dec. 1983, pp.72-77.

47. W. Buxton and B. Myers, "A Study in Two-Handed Input," *Proc. SIGCHI, Human Factors in Computing Systems*, ACM, New York, Apr. 1986, pp.321-326.

48. R. Stallman, D. Weinreb, and D. Moon, Lisp Machine Inc., *Lisp Machine Windows System Manual*, Culver City, Calif. 1983.

49. Apollo Computer Inc., "Window Manager," Chelmsford, Mass., 1981.

**Brad A. Myers** is a research computer scientist at Carnegie Mellon University. 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 interface management systems (UIMSs), user interfaces, programming by example, visual programming, interaction techniques, window management, programming environments, debugging, and graphics.

Myers received a PhD in computer science at the University of Toronto, and MS and BSc degrees from the Massachusetts Institute of Technology. While at MIT, he was a research intern at Xerox PARC. He is a member of SIGGRAPH, SIGCHI, ACM, and the Computer Society of the IEEE.

Myers' address is Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213-3890.