Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical User Interface

D. AUSTIN HENDERSON, JR., and STUART K. CARD
Xerox Palo Alto Research Center

A key constraint on the effectiveness of window-based human–computer interfaces is that the display screen is too small for many applications. This results in "window thrashing," in which the user must expend considerable effort to keep desired windows visible. *Rooms* is a window manager that overcomes small screen size by exploiting the statistics of window access, dividing the user's workspace into a suite of virtual workspaces with transitions among them. Mechanisms are described for solving the problems of navigation and simultaneous access to separated information that arise from multiple workspaces.

Categories and Subject Descriptors: D.4.2 [**Operating Systems**]: Storage Management—*virtual memory*; H.1.2 [**Models and Principles**]: User/Machine Systems—*human factors*; *human information processing*; I.3.6 [**Computer Graphics**]: Methodology and Technique—*ergonomics*; *interaction techniques*

General Terms: Design, Human Factors, Theory

Additional Key Words and Phrases: Bounded locality interval, desktop, locality set, project views, resource contention, Rooms, virtual workspace windows, window manager, working set

1. INTRODUCTION

The small size of computer screens is a more serious impediment for window-based workstations than is often appreciated. This paper presents a window manager design that effectively enlarges the user's screen. The design is based on an analysis of window usage.

Many potential knowledge-intensive computer applications require that the user interact with a moderately large number of objects. For example, paper materials, when used for writing a paper, may easily fill a dining-room table (Figure 1). Furthermore people tend to switch back and forth between parts of a project and among different activities [2, 21]. For example, a person writing a paper on a computer workstation may nonetheless read his or her electronic mail daily, answer letters, perform housekeeping on the computer files, and consult

This research was supported in part by NASA Ames under grant NAG 2-269.
Authors' address: Intelligent Systems Laboratory, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304.
Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1987 ACM 0730-0301/86/0700-0211 $00.75

Fig. 1. Dining-room table being used to manage a complex task by spreading out parts of the task in space so each part is quickly accessible. The large space simplifies the task.

with students. Each of these tasks has its own objects, such as electronic messages and file browsers, for the user to interact with. The result is many pieces of information for a user to look at, manipulate, and track.

When tasks are done with paper, current information is usually managed using a two-dimensional space in the form of a desk or often a dining table, on which papers are grouped and arranged meaningfully. This allows information needed for a task to be placed and ordered in temporary arrangements without the difficult and expensive effort of assigning formal codes or names. The visual availability of the papers in arrangement provides memory cues that organize and substantially ease the task. This natural use of space has been sought for computer systems in the desktop metaphor interface and its variants [l, 4, 24]. Unfortunately, any straightforward attempt to use computer display space in this fashion immediately confronts the problem that the display screen is often too small.

Computer displays are *much* smaller than desks or tables. Figure 2 compares the area outlines of different computer displays with those of a desktop and a dining table. A standard office desk has the area of 22 IBM PC screens, 46 Macintosh screens, or even 10 of the large 19-inch Xerox 1186 or Sun-3 screens. A dining table is the size of 57 PC screens, 119 Macintosh screens, or 27 19-inch screens. And, if one were to include the effects of resolution, gray scale, and color in the computation, the comparison would be even more extreme.

A number of techniques have been proposed for overcoming the small-screen problem. These can be roughly divided into four categories: (1) alternating screen

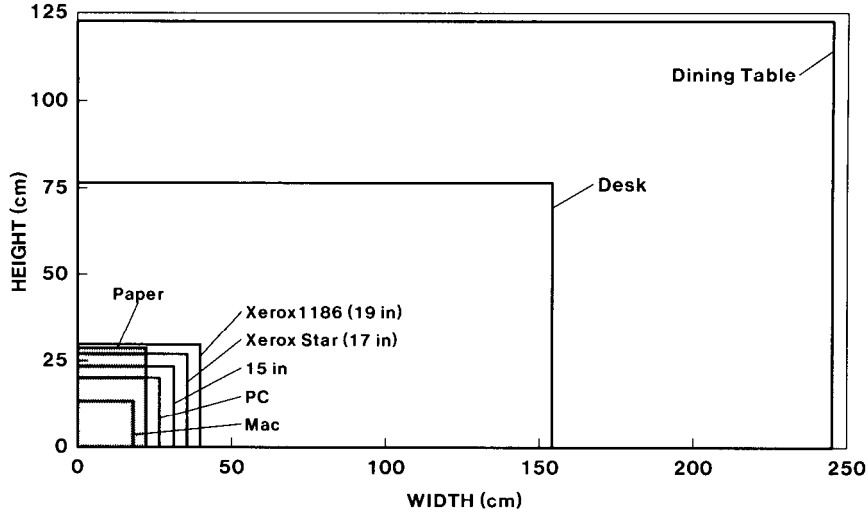


Fig. 2. Superimposed outlines of different workspaces. The crosshatched area shows the size of an 8½-by-11-inch page. Desks and dining tables are very much larger than even large displays. If display resolution were taken into account, the comparisons would be much more extreme.

usage, (2) distorted views, (3) large virtual workspaces, and (4) multiple virtual workspaces.

Alternating screen usage. The user can simply switch the allocation of screen space from one application to another. The Xerox Star [24], for example, provides for storing documents in file drawers to be fetched and reopened as needed. The original Macintosh was a more extreme case, allowing only one application to be on the screen at a time.

Distorted views. Another way of gaining space is to distort the objects in the workspace. One of the oldest techniques for doing this, first appearing in Smalltalk, is the use of icons [23]: Windows are shrunk to small pictures that remind the user of the original window. Overlapping windows, also derived from Smalltalk [14], can be considered a distorting technique: Windows are allowed to cover each other leaving only a portion to remind the user of what lies behind. Icons and overlapping windows are probably responsible for making the electronic desktop metaphor possible at all. But, as Figure 3 shows, in many applications they are not enough. When the need for screen space outstrips the space available, overlapping windows can create an "electronic messy desk" in which the windows interfere with one another. More recently, "fish-eye" distortions have been explored by Furness [13] and Spence and Apperly [25]. The idea is to force all of the objects to fit into the screen space by allocating space to objects on the basis of their intrinsic importance and the user's current focus of attention. Parts of some objects may be clipped or their dimensions distorted to cause them to fit. The Boxer system [8] is a different sort of distorted-view system. A spatial box metaphor is used to represent many semantic notions of containment. The boxes

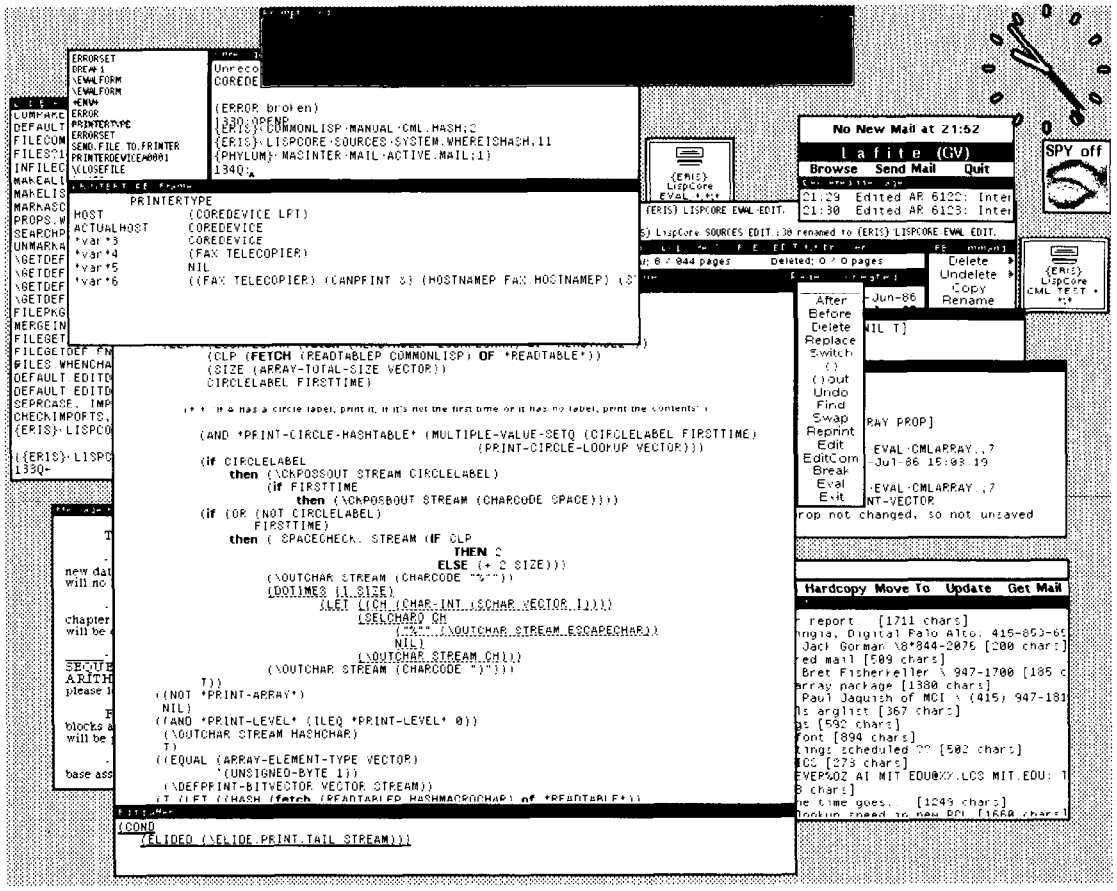


Fig. 3. Overlapped window display. Since the workspace is small, it is easily overloaded and becomes a cluttered desk.

are nested in a hierarchy. Individual boxes can either appear expanded or shrunk to a symbol, depending on where the user is in the hierarchy.

Large virtual workspaces. Instead of forcing all of the objects to fit on the screen, another technique is to arrange them in a single large virtual workspace much larger than the screen. The screen is treated as a movable viewport onto this space. Sketchpad [26] was one of the earliest graphical programs to use this technique. Many other systems have since developed their own versions. In Dataland [1], color pictorial and textual data are arranged in two dimensions. The user has three screens, one for an overview of the whole space, one for a detailed view of some portion of the space, and one touch screen for control. The user can translate or zoom his or her detailed view. As the user zooms closer to the data, more detail is revealed. In each of these systems, the data are passive—the user cannot interact with the data other than to view them. By contrast, the Cedar Whiteboard system [9] also provides translation and zooming over a large workspace, but individual data elements can lead to opening application windows. At the extreme of the virtual workspace systems are head-mounted displays (e.g., the NASA helmet [12]), which monitor user head and body movements to give the user a complete simulated 3D space.

Multiple virtual workspaces. The simulation of a single large workspace is natural, but carries over to the system some strong constraints of physical space—only a limited number of things can be adjacent to any object, for example, and the space required for the objects and their shapes puts strong constraints on how the space can be arranged and how densely it can be packed. An alternative to a single virtual workspace is to have multiple virtual workspaces, that is, geometrically oriented workspaces linked to each other, perhaps nonspatially. An example is Smalltalk Projects [14]. Each project contains a number of views and, when active, takes up the whole screen. Projects are arranged hierarchically, with subprojects represented in their parents as windows through which the projects can be entered: these windows are called ProjectViews, or (informally) doors. More direct access to all the projects can be created through browsers, which permit referencing by name all the projects at once. In the CCA system [1, 16] (a descendent of Dataland), whenever a user zooms close enough to a port, he or she is swept through into a subworkspace. Like Smalltalk Projects these subworkspaces are arranged hierarchically. A third example of multiple virtual workspaces is the Cedar programming environment multiple desktop overview [20]: 16 desktops are shown in miniature. The user selects which desktop to enter from this overview. Finally, in Chan's UNIX[1]-based Room[2] system [4], each workspace contains a set of collected icons that provide actions appropriate for carrying out a particular task: either to move to another workspace (doors) or to start a new process for the task the icon specifies. Moving to another workspace provides access to other icons.

[1] UNIX is a trademark of AT&T Bell Laboratories.

[2] We did not discover the existence of the similarly named system "Room" (Chan's Master's thesis at the University of Waterloo [4]) until after we had publicly demonstrated our Rooms system at the AAAI conference in August 1986. To avoid confusion in this paper, we refer to Chan's system as "Chan's Room system."

A more complex organization for multiple virtual workspaces than the tree-oriented systems described above is the Feiner et al. electronic book [11]. Pages in the book (each separate workspaces) are organized into subchapters and then chapters. Pages may contain pictures, any of which can be active in preset ways. Among other things, this provides single-action access to other pages of the book. Pictures can be used in different scales, another variant on the distortion technique.

At the extreme of increasing complexity of structure are the hypertext systems, characterized by small, often textual, networks of workspaces connected with arbitrary patterns of typed links. The earliest of these was NLS [10]. More recent examples are PROMIS [17], ZOG [22], and NoteCards [15]. PROMIS and ZOG display a single node at once; NLS provides access to a subtree of nodes, screen space permitting; and NoteCards provides access to any arbitrary set of nodes. Motion among the nodes is by traversing links.

The above techniques make progress toward solving the small-screen problem, but precipitate two new problems, which arise when not all the information is visible on the screen at once: (1) the problem of navigation (how to find objects in the workspace) and (2) the problem of arranging simultaneous access to separated information.

The problem of navigation. This is the user's problem of finding the way to information without getting lost. In large virtual workspaces with a strong physical model (Dataland, Whiteboard, etc.), the spatial analogy is a powerful space organizer. Thus these systems base much of their navigation on translating and zooming and may provide both global and local views [1]. As in unfamiliar cities, however, it may still be possible to get lost or not find what one is looking for. In systems with multiple virtual spaces (e.g., Cedar, Smalltalk Projects), an overview can also be provided if the number of virtual workspaces is not too large. In hypertext systems (e.g., Electronic Book, NoteCards) the task is harder. Here, the navigation issue is not aided by a strong physical model, and the separate nodes are related by a tangle of links. Visual presentation of the links may be more confusing than enlightening. One solution is presentations of local connectivity (Electronic Book, NoteCards); another is browser presentations of nodes (Smalltalk Project Browser, Electronic Book chapter structure, NoteCards Browser).

Simultaneous access to separated information. It is often necessary to bring together two or more pieces of information from separated parts of the workspace. In fact, the same piece of information may even be logically associated with more than one part of the workspace. In some systems, particularly those with a strong spatial model (e.g., Dataland), this presents the difficulty of having to destroy the old organization to gain the new. Of course, one can make copies, but copies have coordination difficulties if the information can change. Distortion (fish-eye views, icons, overlapped windows) provides another solution: The arrangement remains fixed, while the distortion can change to make the desired information simultaneously accessible. Sharing information (the same picture can be used on different pages in the Electronic Book) is another solution, one requiring more sophisticated information structuring. In the extreme, hypertext systems (e.g.,

NLS, NoteCards) provide for multiple linkings, which, in turn, provide multiple clusterings through separate views.

In the system we propose, our object is to prototype a workstation window manager that allows users to operate with larger collections of objects. Although a number of systems have developed techniques for mitigating the small-screen problem, no generally accepted framework for understanding the problem itself has yet emerged. As part of the process of advancing these techniques, we wish to begin building an understanding of the problem to aid us.

2. ANALYSIS OF DESKTOP INFORMATION USE

Instead of just developing another system to mitigate the small-screen problem, it is useful to construct some analytical understanding of the key constraints acting upon desktop information use. From this understanding we can gain abstractions with which to organize the design space and gain insight into promising regions of the design space for siting a design.

Basically, our analysis is that (1) the high overhead of moving and reshaping windows that users must often suffer with overlapped window systems derives from a severe screen-space resource contention lying just below the surface, but that (2) we can partially overcome this resource contention by designing a virtual workspace manager that exploits the statistics of window reference.

2.1 The Small-Screen Problem

Informally, the resource contention can be described by the following *gedanken* experiment: Imagine the task of writing a paper using a dining-room table. Drafts, figures, references, outlines, and notes can be spread on the table in a way that makes them easily accessible as the writer proceeds. Imagine the same task now done on an office desk. There is still considerable workspace, but perhaps some of the information must be piled together, necessitating occasional flipping through piles. Now imagine the task on a very tiny desk. Much of the writer's time must be devoted to thrashing about searching through papers: New papers dredged up top will cover other papers still in active use, and these, in turn, will need to be dredged up and will cover other papers in use. In addition to the immediate time consequence of thrashing, the ensuing chaos will tend to alter the task itself, pushing the writer toward more formal methods of accessing information, such as file folders and note cards, which have their own overheads: time to categorize, memory for categories, and conceptual ambiguities.

More formally, an analysis of resource contention leading to "thrashing" behavior is available from the study of virtual-memory operating systems [7], and we can apply some of its features to the small-screen problem. In a virtual-memory operating system, programs are run in a large, virtual-address space. This virtual-address space resides on some secondary storage medium, such as a rotating disk, much larger (and also much slower) than the physical address space available in the main computer. When reference is made to a page in virtual memory not actually resident in the computer's main memory, a "page fault" occurs: A page in physical memory is written back to disk if it has changed, and the referenced virtual-memory page is copied to main memory in the vacated slot.

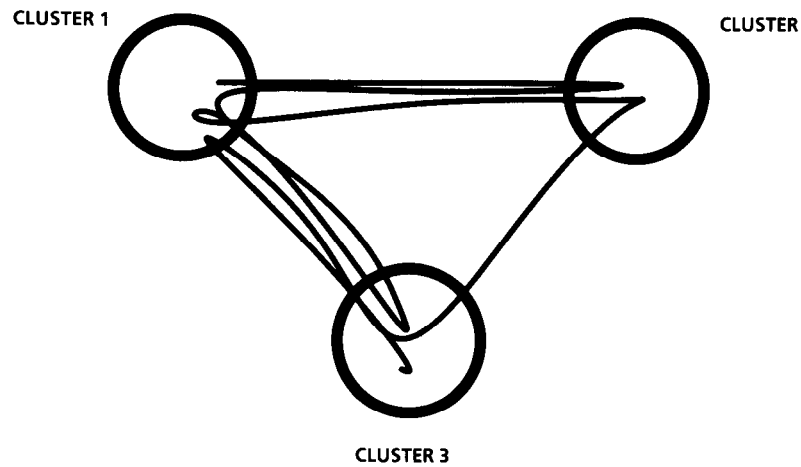


Fig. 4. Locality sets and possible transitions between them. Each circle represents a cluster of page locations that tend to be referenced together in time. The line represents the locus of program control as it transitions back and forth between such clusters. Programs spend most of their time in clusters of page references, represented by the circles, and relatively little time (from 2 to 5 percent) in transitions between clusters, yet typically half of the page faults occur during the transitions. (After Madison [19, p. 26].)

Two principal factors determine the success of this scheme: (1) the size of the main memory available relative to the size of the program and (2) the statistical distribution of the program's references to virtual memory. With respect to the latter, a program that exhibits *locality of reference*, that is, a program whose references to virtual-memory page locations cluster together in time, causes fewer page faults and hence requires much less time to run (or much less main memory to run at a given speed) than one whose references are randomly distributed. Fortunately, most programs progress in distinct *locality phases* [7]: That is, most programs progress by making frequent references to a small set of virtual-memory locations (called a *locality set*) followed by transition to another small set of virtual-memory locations (Figure 4). Page faults tend to be relatively sparse within phases, but dense within transitions. For example, Kahn measured several programs [18, cited in 7] and found that the programs in his sample spent 98 percent of their time within some phase. But during the 2 percent of the time they were in transition, 40–50 percent of the page faults occurred [7, p. 72].

Reasons for locality in program reference behavior are not difficult to find. First, programs typically consist of blocks of memory that are executed more or less sequentially. Second, programs often have loops that iterate over the same locations. Finally, programs usually involve bursts of computations on related variables or data structures.

The reference characteristics of programs, then, essentially derive from two factors: (1) the sequential storage structure of most program code as assembled by a compiler and (2) the way in which references to the same variables tend to be clustered in time. It is important to note that locality of reference is strongly

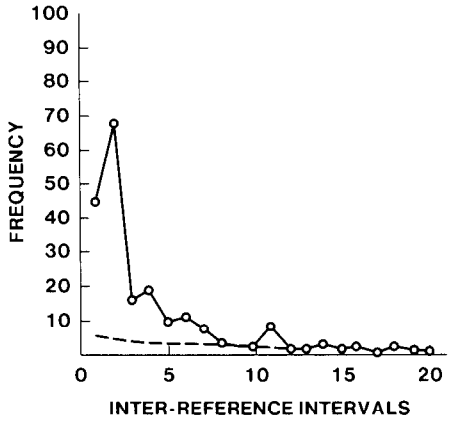


Fig. 5. Window interreference interval distribution. The graph plots the frequency with which different numbers of window references intervening between references to the same window are observed. For example, there were 10 instances in which the same window appeared as the fifth window reference after it had previously appeared. The graph gives evidence for considerable locality in window references. If the references were uniformly random, the distribution would follow the dotted line. The actual distribution is heavily skewed toward short interreference intervals indicating locality. (From Card, Pavel, and Farrell [3].)

evident, even when this latter case of reference to variables is considered alone [7].

For virtual-memory operating systems, the size of main memory relative to the size of the virtual memory and the high cost of references to secondary memory are *key constraints* that drive the performance of the system. Main memory is a limited resource that is in contention. Operating-system algorithms attempt to exploit locality of reference in programs to reduce memory contention. Since memory contention is often a performance driver, reducing it can have a major impact on system performance as a whole.

For windows (as well as for paper-laden desks), the size of the available workspace relative to the needs of the task to be done is a key constraint that drives user performance on the task. Small display screens or desks are limited resources that are in contention. But, if users exhibit locality of window reference and a system can make explicit use of it, we can likely make a major improvement in window manipulation overheads that derive from the small-screen problem.

2.2 Window Locality Sets

Evidence for locality of window reference seems clear enough from protocols we have been able to examine of users interacting with an overlapped window system. In the first place we have informally observed such behavior: Users are often seen to use a group of windows for a while, then delete or shrink most of them, and then begin building another set. More systematically, the distribution of window interreference intervals (the number of window references between two references to the same window) is shifted heavily to the low end (compared with what would be expected for random window referencing), indicating that the user in this protocol often went back and forth among a small set (five or fewer) of windows (see Figure 5).

Other ways of looking at user window behavior suggest a similar conclusion. We can identify approximately the locality set of windows in active use at a given time by computing Denning's *working set* [5–7]. The working set consists of those windows referenced in the last T references. Figure 6 shows this metric computed for a user programming Interlisp-D on the basis of those window events detectable

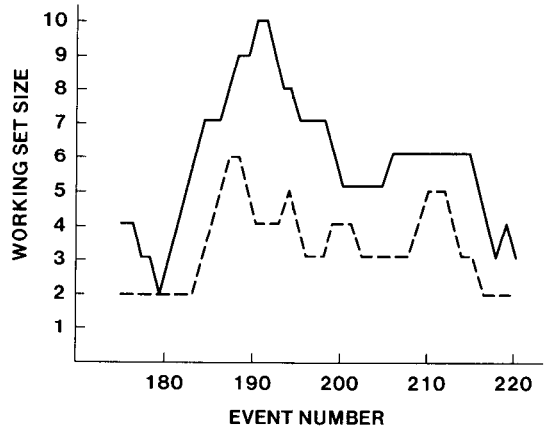


Fig. 6. Computation of Denning's working set for window references made by a user reading mail and programming in Interlisp-D: ----, $T = 6$; ———, $T = 14$. Computation is based on detectable system events. The waxing and waning of the number of windows in the computation interval T is clearly visible, but this way of estimating window locality sets does not show well the boundaries of the phase transitions. (From Card, Pavel, and Farrell [3].)

by the system. The figure shows a window working set varying between 2 and 10 windows for the fragment of behavior examined.

However, a better metric for our purpose is Madison's *bounded locality interval* (*BLI*) [19]. This metric overcomes two problems with the classical working-set measure: (1) that it depends on an arbitrary parameter T, the reference interval, and, more important, (2) that the interval size T is fixed, with the consequence that the boundaries between phases are not well defined [19]. The bounded locality interval is based on the top elements of an LRU (least recently used) stack that have all been referenced at least once since their formation (see [19]). The BLI metric has been shown by Madison to correspond well to intuitive notions of a phase. Figure 7 shows the BLI metric computed over 60 minutes of behavior on the Interlisp-D system. The computation indicates the existence of from zero to five windows in a locality set at any time for this fragment of user behavior. (Had the user referenced windows randomly, Figure 7 would be blank.)

These four indicators—simple observation, interreference interval, Denning's working set, and Madison's bounded locality interval—cannot yet be considered definitive pending further studies and refinements, but they do at this point all suggest in unison that users exhibit locality of window reference and that, therefore, as in the case of virtual-memory operating systems, mechanisms that exploit this statistical property of user behavior might be developed.

2.3 Tasks and Tools

It is not difficult to suggest reasons for the locality-of-window-reference behavior observed above. When there is some *task* to be done, such as reading mail, writing a paper, or creating a program, the user gathers a number of *tools* for doing it. In

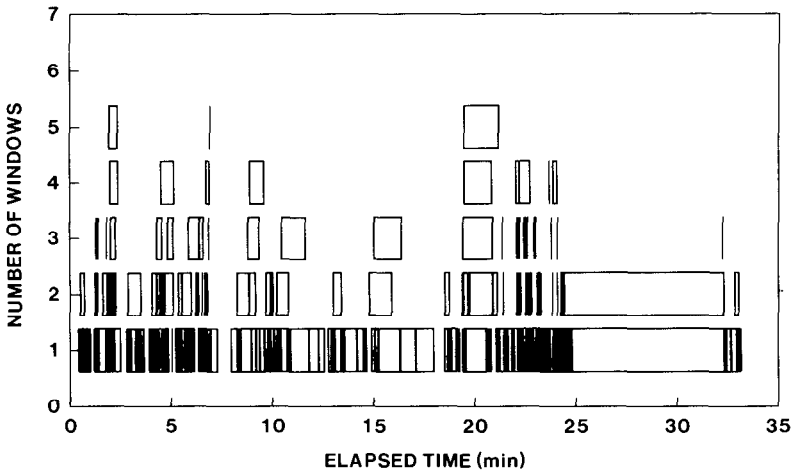


Fig. 7. Computation of the Madison's bounded locality interval (BLI) metric. The computation is based on a user programming Interlisp-D. In order to capture part of the user's visual reference to windows, windows in the enviornment have been programmed to "gray out," becoming barely readable after 10 s. The windows are instantly restored if the user selects them with the mouse or the system uses them. Each rectangle is a locality set plotted according to the number of windows in the set. The abscissa is time in milliseconds. If window references did not exhibit locality, there would be no rectangles. With this technique the boundaries of the window locality sets are easier to see. The technique shows promise, with refinement, for being able to describe complicated patterns of window use.

most window systems, these tools are each embedded in a window (e.g., mail-browser windows for mail; text-editor and file-browser windows for paper writing; program editors, debuggers, and measuring instruments for programming). As the user moves back and forth among these tools while doing the task, the tool windows form a locality set.

A task, from the point of view of the user, will therefore tend to correspond to a phase from the point of view of window reference statistics. Task switching will correspond to a transition, and we expect it to lead to heavy window faulting. The design of the Rooms system is based on the notion that, by giving the user an interface mechanism for letting the system know he or she is switching tasks, it can anticipate the set of tools/windows the user will reference and thus preload them together in a tiny fraction of the time the user would have required to open, close, and move windows or expand and shrink icons. A further benefit is that the set of windows preloaded on the screen will cue the user and help reestablish the mental context for the task. (We might even say that knowledge faulting in the user is thereby reduced.)

3. DESIGN OF THE ROOMS SYSTEM

The window-management system we describe provides the user with a suite of roughly screen-sized workspaces called *Rooms*. The overall scheme is shown schematically in Figure 8. Each Room contains a set of window *Placements*, each of which indicates a window, a shape, and other presentation information

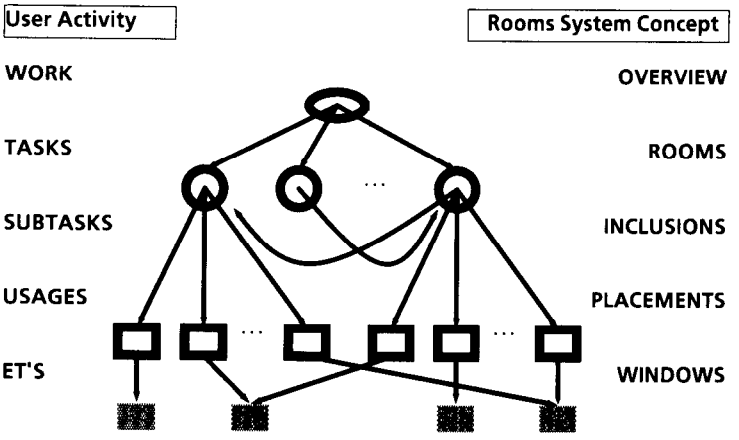


Fig. 8. Schematic structure of the Rooms system. Users' work is distributed among several Rooms, each Room containing a main task. Rooms can be included in other Rooms. Each Room contains a set of Placements. Placements indicate windows and Room-dependent presentation information for the window.

(i.e., how a given window is to appear in a particular Room). A key feature of Placements is that two Placements may refer to the same window, allowing windows to be shared among Rooms. Rooms may also be included in other Rooms as a mechanism for allowing the sharing of groups of windows among workspaces. Our system is implemented in Interlisp-D and uses the Interlisp-D window package for basic window manipulations.

Figure 9 shows two typical Rooms. The Room in Figure 9a has been laid out with mail-browsing windows and a mail-reader window. It has a prompt window for messages (the long black rectangle), two icons with controls for the mail system, a command typescript window for typing LISP functions, a clock, and *Doors* linking this Room directly with others. If the user selects the Door labeled PROG with the mouse, then the screen changes to the Room in Figure 9b; that is, the user has the illusion of moving to Room PROG. This latter Room is set up for doing programming and at the time of entering contains windows in which editing with the LISP structure editor is in progress. It also contains a *Back Door* (the Door in Figure 9b in reverse video) showing the Room from which the user came (and to which the user could return by means of a single mouse selection). To help the user navigate, the system has an *Overview* (Figure 10) that displays miniature versions of all the Rooms in the total user workspace. Any Room can be entered from the Overview, and indeed Placements can be copied, deleted, moved, shaped, and examined at full size from this Overview. Figure 11 shows the set of Rooms and direct Door links of an actual user workspace. The diagram has an obvious similarity to the phase and transition diagram of Figure 4.

The design for Rooms is based on our analysis of the key constraints on user window behavior as developed in the previous section. Since on present evidence much of the user housekeeping for an overlapped window system derives from contention for limited screen space, the design gives the user more virtual space. Since we expect the use of that space to be characterized by phases and transitions

organized around tasks, this total virtual workspace is divided into multiple virtual subworkspaces through Doors (or the Overview). Our basic analysis, therefore, has helped suggest a promising position in the design space at which to site a design. But the design decisions to which we are thereby led have further entailments of their own. The design for Rooms reflects solutions that arise not only from the basic analysis, but also from these entailments—issues that may have little relation to the original problem that Rooms was designed to solve, but whose resolution is necessary for the system to be viable. As is the case for all systems in which information is not all visible on the screen at once, the Rooms system must face entailed issues that derive from (1) problems of navigation and (2) problems of arranging simultaneous access to separated information. A third group of entailed issues focuses around (3) simple user tailoring of system appearance and behavior. The major entailed design problems and their solutions are summarized in Table I. It is simplest to begin with problems of simultaneous access.

3.1 Simultaneous Access to Separated Information

Some windows, such as the invocation of a text editor on a particular file, are strongly associated with a particular task. Others, such as a clock or the command typescript window, are relatively independent of different tasks. Each Room tends to be organized around some dominant task, such as writing a paper or reading the mail. The windows contained in a Room provide the tools for the task and, indirectly, the material to which the tool is being applied (e.g., a window containing a text editor opened to a particular file). But tools and tasks can be combined in different ways. A single-purpose mail Room might have just mail-transport tools, mail files, and mail sorters in it, whereas a project Room might have a mail reader, a text-editor window, and a programming editor. So, although it is easy to recognize the global need for some sort of simultaneous access to the same windows across different Rooms, there are, in fact, a number of specific cases to be sorted out before sharing can actually be accomplished. These can be classified as (1) sharing tools across tasks, (2) workspace-dependent window presentations, (3) sharing collections of tools across tasks, and (4) carrying tools to another workspace.

3.1.1 *Sharing Tools across Tasks*

ISSUE 1 [*Multiple instances of windows*]. *Some windows need to appear in more than one workspace.*

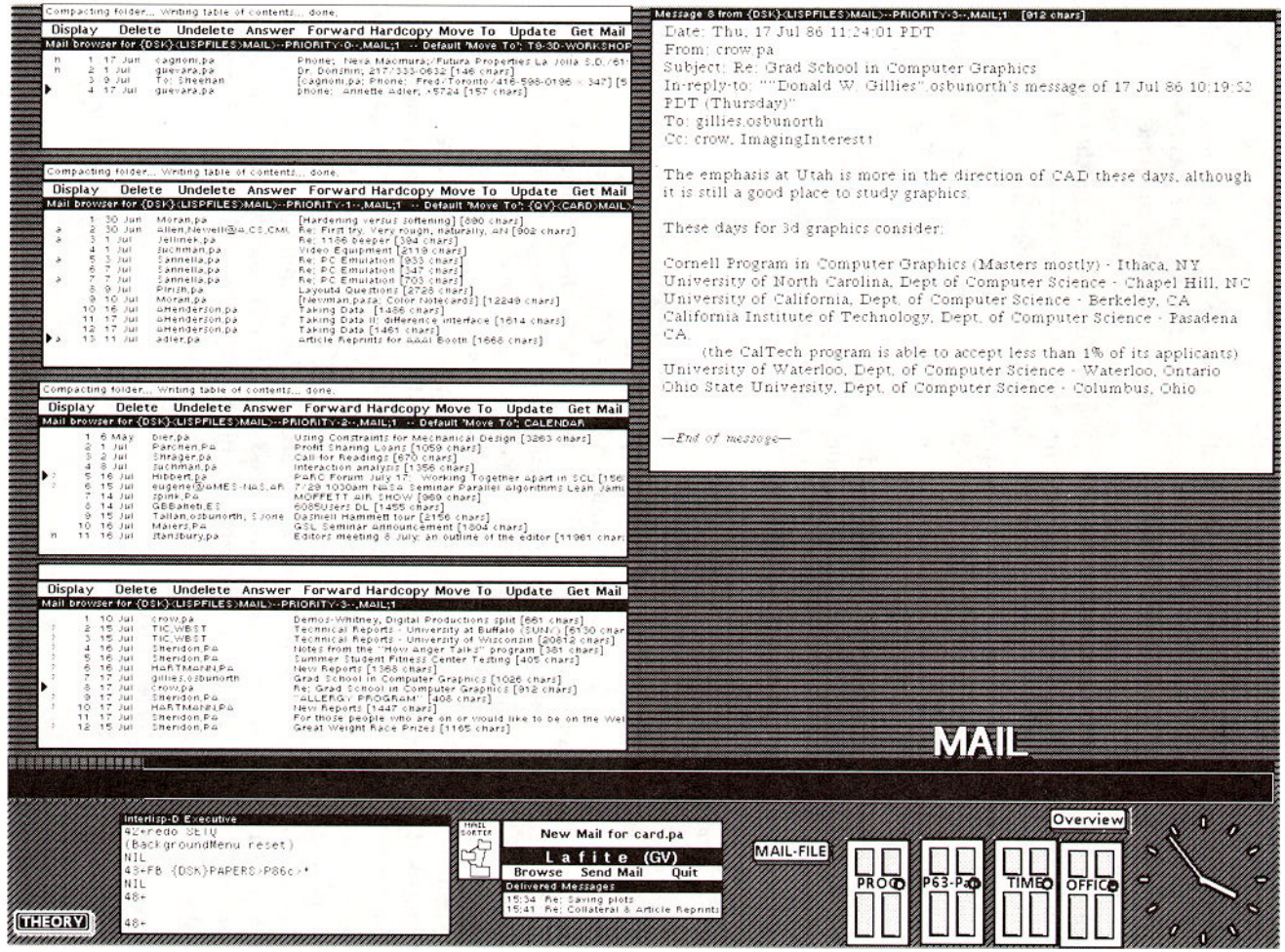
DESIGN SOLUTION. *Multiple Placements of a window.*

The desire to have versions of the same window appearing in more than one Room forces us to the abstraction of a *Placement*. A Placement is a window together with location and presentation information.

Placement = Window + LocationInRoom + PresentationAttributes.

A Placement generalizes the concept of a window, separating the tool aspects of it (the fact that a particular set of editing commands work inside it) from its appearance on the screen.

(a)


Compacting folder... Writing table of contents... done.
Display Delete Undelete Answer Forward Hardcopy Move To Update Get Mail
Mail browser for {DSK}<LISPFILES>MAIL>--PRIORITY-0-.;MAIL;1 -- Default 'Move To': T9-3D-WORK:SHOP
Compacting folder... Writing table of contents... done.
Display Delete Undelete Answer Forward Hardcopy Move To Update Get Mail
Mail browser for {DSK}<LISPFILES>MAIL>--PRIORITY-1-.;MAIL;1 -- Default 'Move To': {QV}<CARD>MAIL>
Compacting folder... Writing table of contents... done.
Display Delete Undelete Answer Forward Hardcopy Move To Update Get Mail
Mail browser for {DSK}<LISPFILES>MAIL>--PRIORITY-2-.;MAIL;1 -- Default 'Move To': CALENDAR
Display Delete Undelete Answer Forward Hardcopy Move To Update Get Mail
Mail browser for {DSK}<LISPFILES>MAIL>--PRIORITY-3-.;MAIL;1
Message 8 from {DSK}<LISPFILES>MAIL>--PRIORITY-3-.;MAIL;1 [812 chars]
Date: Thu, 17 Jul 86 11:24:01 PDT
From: crow.pa
Subject: Re: Grad School in Computer Graphics
In-reply-to: ""Donald W. Gillies".osbunorth's message of 17 Jul 86 10:19:52 PDT (Thursday)"
To: gillies.osbunorth
Cc: crow, ImagingInterest↑
The emphasis at Utah is more in the direction of CAD these days, although it is still a good place to study graphics.
These days for 3d graphics consider:
Cornell Program in Computer Graphics (Masters mostly) - Ithaca, NY
University of North Carolina, Dept. of Computer Science - Chapel Hill, NC
University of California, Dept. of Computer Science - Berkeley, CA
California Institute of Technology, Dept. of Computer Science - Pasadena CA.
(the CalTech program is able to accept less than 1% of its applicants)
University of Waterloo, Dept. of Computer Science - Waterloo, Ontario
Ohio State University, Dept. of Computer Science - Columbus, Ohio
—End of message—
MAIL
Interlisp-D Executive
New Mail for card.pa
Lafite (GV)
Browse Send Mail Quit
Delivered Messages
15:34 Re: Saving plots
15:41 Re: Collateral & Article Reprints
MAIL-FILE
Overview
PROG
P63-Pa
TIMES
OFFICE
THEORY

(b)

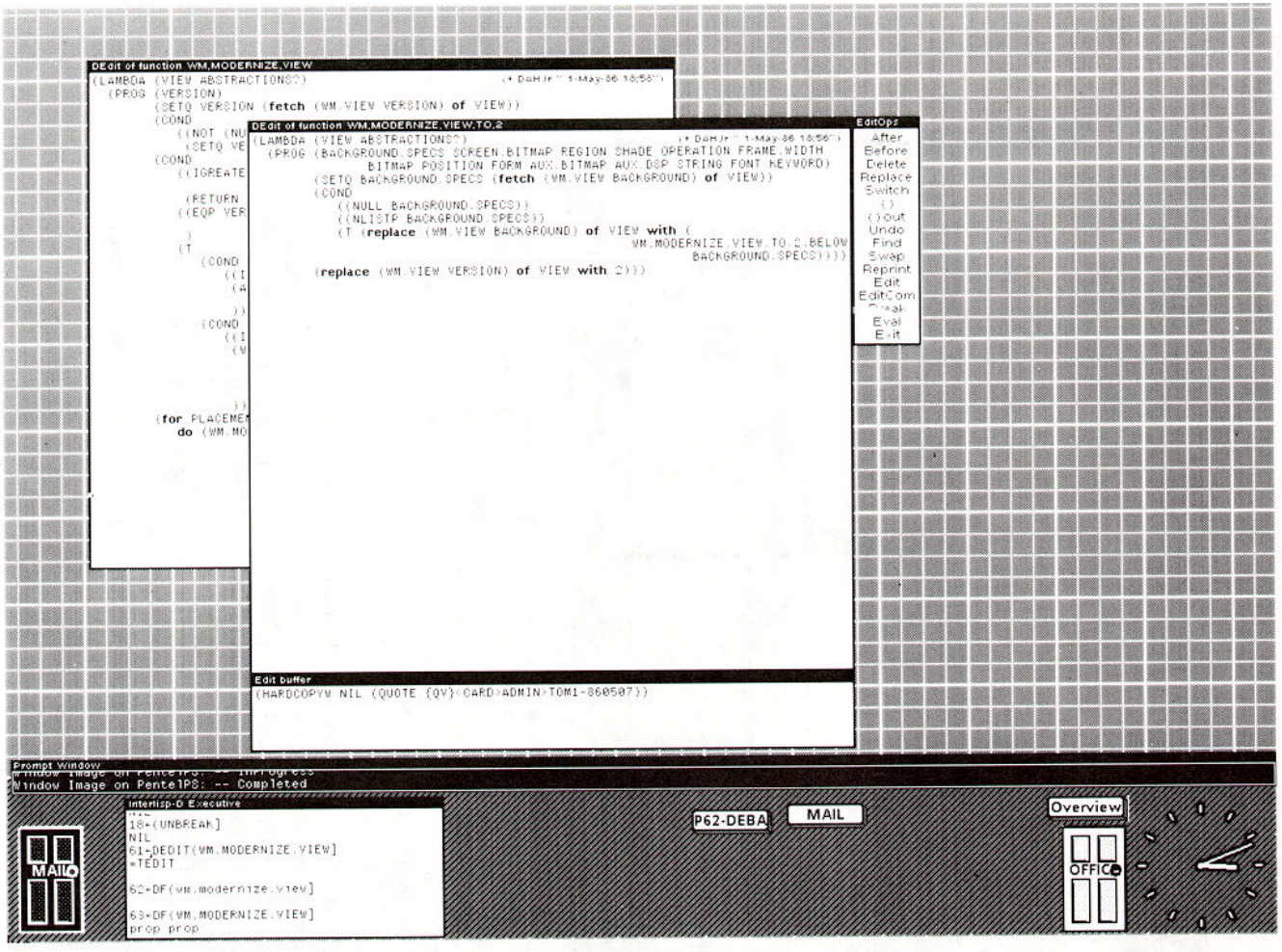


Fig. 9. Two examples of Rooms. Room (a) is used for reading mail; Room (b) for programming. In these Rooms both the panel door figures and the button-shaped figures (e.g., the one marked "Overview") are doors.

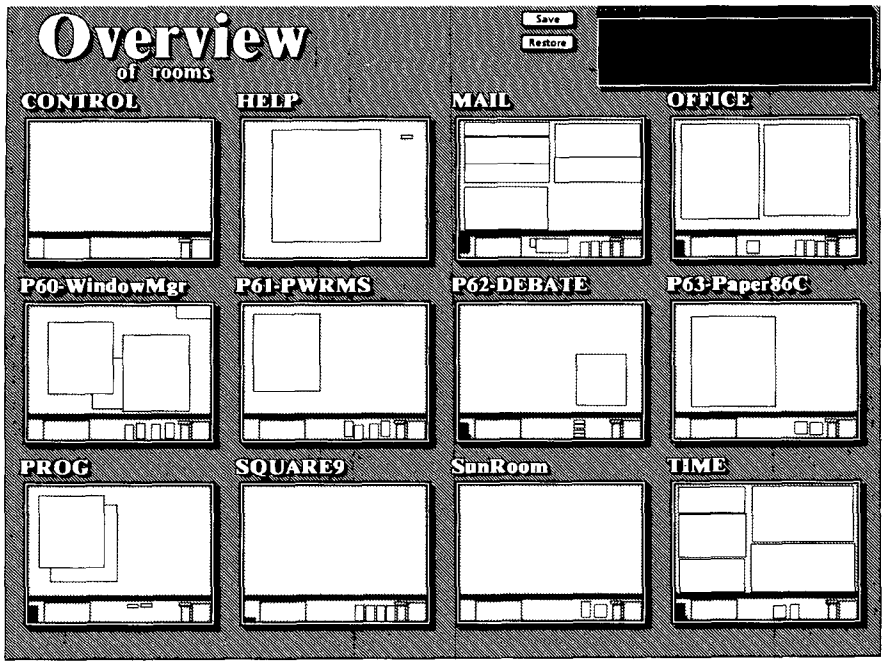


Fig. 10. Overview. The Overview contains pictograms of the Rooms arranged alphabetically. It also contains a message window for communicating with the user and buttons for saving and restoring the set of Rooms. This Overview shows a CONTROL Room that is included in every Room except the HELP Room. Windows contained in a Room because they are part of an included Room are rendered in gray. **EXPAND**ing the window in the HELP Room provides the user with a one-page illustrated system manual.

3.1.2 *Workspace-Dependent Window Presentations.* Windows shared by different workspaces may need to have different locations in different workspaces. This is immediately obvious in application, but would not be possible if the Rooms system merely contained lists of the windows present in each Room.

ISSUE 2 [*Workspace-dependent shared-window positions*]. *Shared windows need to have independent positions in different workspaces. Repositioning a shared window in one workspace should not affect its position in another workspace.*

DESIGN SOLUTION. *Position is part of a Placement, not a window.*

The independent location problem is also solved by the Placement mechanism. A Placement contains an x, y position for the bottom left corner of a particular window in the particular Room.

It is possible to go beyond the simple location of shared windows in the different Rooms to other aspects of presentation. For example, it may be desirable to have a text-editor window be large in one Room, but small in another. Or we may want the text-editor window to be squarish in one window, but tall and thin in another so as to fit into a differently arranged space. Or we may wish a window to have drop shadows in one Room, but not in another.

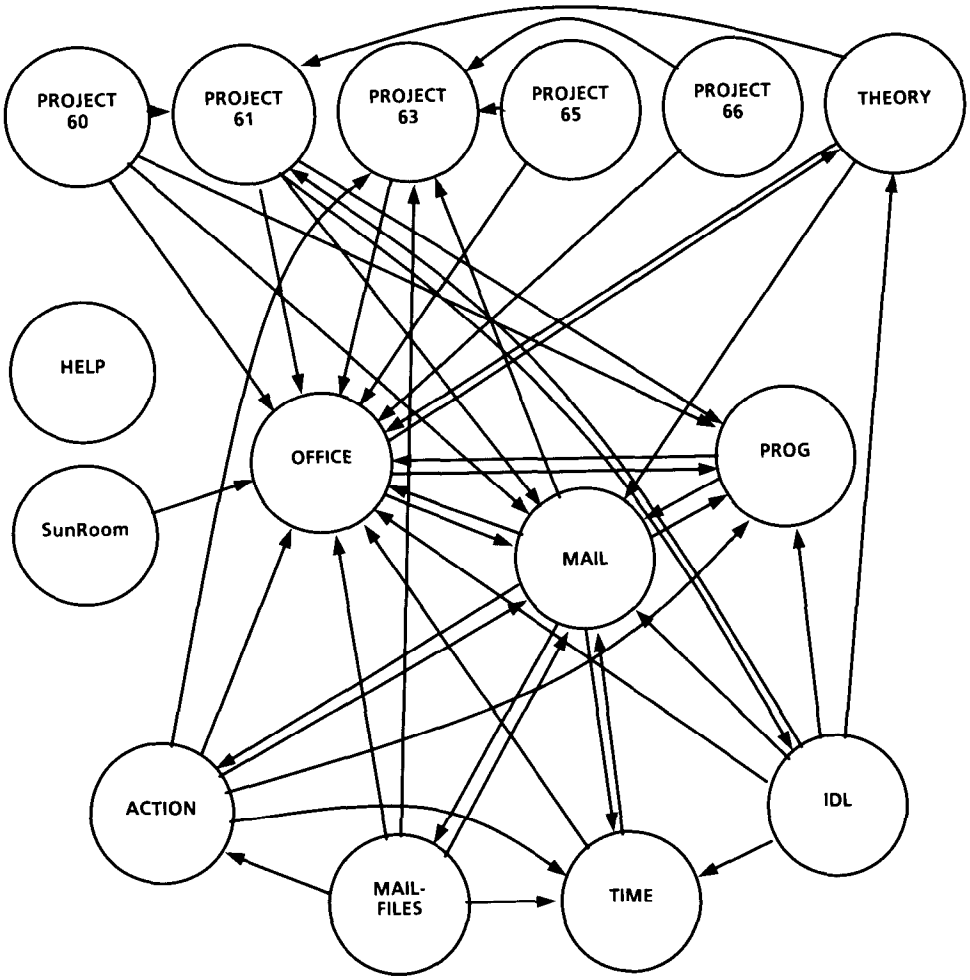


Fig. 11. Transitions among Rooms in a user's workspace. The diagram shows which Rooms have doors leading to other Rooms. It is an analog of Figure 4. Transitions through the Overview or pop-up menu are not shown.

ISSUE 3 [*Workspace-dependent shared-window presentation*]. *The shape, size, icon shrink location, drop-shadow attributes, and other aspects of the presentation of shared windows need to vary according to which Room the windows are in.*

DESIGN SOLUTION. *Presentation attributes are properties of Placements, not windows.*

Fortunately, the Placement mechanism is again a good solution to this issue. The Placement for each window in a Room contains slots for the presentation dimensions on which windows can vary.

3.1.3 *Sharing Collections of Tools across Tasks.* The issues above have derived from cases in which we want the same windows to have location and presentation

Table I. Summary of Design Issues That Arise in Going from the Virtual Workspace Idea to a Usable System

Issues		Design solution
(1) Interface issues of simultaneous access to separated information		
Issue 1	Multiple instances of windows	Placements
Issue 2	Workspace-dependent window locations	Placements
Issue 3	Workspace-dependent window presentations	Placements
Issue 4	Collections of windows	Room inclusion
Issue 5	Bringing windows to other workspaces	Baggage
Issue 6	Keeping windows along with user	Pockets
(2) Interface issues of navigation		
Issue 7	Reversibility of workspace transition	Back Doors
Issue 8	User orientation	(1) Pop-up menu of Rooms (2) Overview (3) Expanding pictograms
Issue 9	Showing workspace connectivity	Wiring diagrams
(3) Interface issues of tailorability		
Issue 11	Rooms redecoration	(1) Maintain normal LISP (2) Persistence of modifications (3) Pop-up menu (4) Overview commands
Issue 12	Unanticipated modifications structure	(1) Editable Rooms data (2) Layout language
Issue 13	Saving/restoring workspaces	Save/restore buttons

aspects that can be different in each workspace. But there are other cases in which just the opposite is true. An example is that we may wish to define a collection of windows to serve as a sort of control panel for many Rooms. Such a collection might contain a command typescript window (where typed-in commands can be evaluated), a clock, indicators for system performance, and Doors to some standard places.

ISSUE 4 [*Collections of windows*]. *Some groups of windows need to be defined as a collection whose location and positional attributes remain constant across workspaces. Changes to any of the windows need to be propagated to all workspaces containing them.*

DESIGN SOLUTION. *Room inclusion.*

Our solution to this problem is to allow Rooms to be included in other Rooms. A control panel is designed by making up a Room with the clock and other useful tools positioned together. We then make this Room an Inclusion of each Room that is to share the collection of windows. Resulting Rooms, when displayed to the user, will contain the combined set of windows. Figure 10 shows a control panel Room marked CONTROL that is contained in another Room. Figures 9a and b contain the windows in this control panel.

3.1.4 *Carrying Tools to Another Workspace.* So far we have not discussed how sharing of windows across workspaces arises. How can the user take a window that is in one workspace and carry it to another?

ISSUE 5 [*Carrying windows to other workspaces*]. *How can a set of windows be copied from one workspace to another?*

DESIGN SOLUTION 1. *Baggage.*

Our solution is to allow the user to carry some windows with him or her as the user transits to another workspace. The metaphor is that the user has Baggage that can be packed full of windows (actually, copies of Placements). The user presses a key while selecting a Door, which puts the user into a mode (with suitable feedback) in which he or she can point to all the windows wanted as Baggage. The Baggage goes through the Door with the user, and the windows assume their former positions, but in the new Room. The user can then reposition the windows in the new Room as desired.

DESIGN SOLUTION 2. *Overview move and copy commands.*

If the user is in the Overview or willing to go to the Overview, then the **MOVE** and **COPY** commands can be used to move or copy Placements rapidly from one Room to another.

Finally, there are applications in which the user wishes to define windows that are present no matter which workspace is used.

ISSUE 6 [*Keeping windows along*]. *In some applications windows need to be associated with the user rather than the workspace.*

DESIGN SOLUTION. *Pockets.*

The user can declare one Room to act as Pockets. This Room will be temporarily included in any Room the user enters. Thus, whichever windows are placed in the user's Pockets will be presented (at the same location and with the same presentation attributes) in all Rooms. A special application is that control panels can be included in a user's Pockets, if the user wants all Rooms to have the same control panel.

3.2 Interface Issues of Navigation

The Rooms system attempts to reduce space contention on the screen by distributing the user's windows into window locality sets in virtual workspaces. But this very fragmentation of the space creates a navigation problem: How can one keep track of the windows no longer visible and find one's way through the Rooms? If this challenge is not met, we shall only have replaced the electronic messy desk with an electronic maze. To keep the overall strategy viable, interface solutions must be found for this problem. Actually, the overall issue of navigation contains several subissues: (1) returning to a Room, (2) general orientation and finding other Rooms, (3) finding windows, and (4) finding which Rooms connect.

3.2.1 *Returning to a Room.* Frequently a user wishes to return from a present Room to the immediately prior Room. This can be a problem because Doors are one-way only. It can be even more of a problem if the user has forgotten the name of the previous Room or even what was being done before an interruption.

ISSUE 7 [*Reversibility of workspace transitions*]. *How can the user return to a previous workspace?*

DESIGN SOLUTION. *Back Doors.*

Each time a user enters a Room, a Door to the previously occupied Room is created and placed at a certain location. This Door is shown in reverse video to indicate that it is a Door back to the previous Room. An example is the reverse-video Door at the bottom left corner of Figure 9b. Further, the Door is destroyed after one use, and no Back Door is created when a Back Door is used to change Rooms. Back Doors reduce the task of returning to a previous Room from a major navigational undertaking to a trivial matter.

Other solutions are possible, but have disadvantages: All Doors could be bidirectional. The problem with this solution is that, when a Door is created in one Room, it would require placement of the Door in both the current visible Room and the indicated invisible Room. Either we would have to be willing to allow the new Door to appear over whatever happened to be in the other Room, or we would have to have an automatic window placement scheme (not a bad idea, but one beyond our current project). Also, Doors back to included Rooms would be conceptually obscure. Another problem with bidirectional Doors is that we want Back Doors to occur whether we entered the Room through a visible Door, through the pop-up menu, or through the Overview. In our system the Back Door is destroyed after one use because the meaning of Back Doors is to help the user return to where he or she just was, and multiple Back Doors would confuse this concept or complicate it with more interface mechanism.

3.2.2 *General Orientation and Finding Other Rooms.* A related problem is that the user can become disoriented. As the number of Rooms increases, the user can find it difficult to remember what Rooms exist. If there were to be a Door from every Room to every other Room, the Doors themselves would soon become unmanageable and consume inordinate amounts of screen space. Yet, without a fully connected topology, the space begins to become complex and difficult to remember—an electronic maze.

ISSUE 8 [*User orientation*]. *How can users remember (or discover) the route to particular workspaces?*

DESIGN SOLUTION 1. *Pop-up menu of Room names.*

The solution from within a Room is to have a pop-up menu that gives the names of the other Rooms (see Figure 12). The menu is a reminder of what Rooms exist, permitting selection of the one the user wants to go to. This solution is cheap in terms of response time, and, just as important, it is always available, even if the user deletes all the windows on the screen. However, it does not remind the user of the contents of the different Rooms.

DESIGN SOLUTION 2. *Overview.*

General orientation is achieved through the Overview. Figure 10 shows the Rooms Overview screen. The main feature of the Overview is a set of Room pictograms, reduced pictures of the Rooms, arranged in alphabetical order. All rooms are displayed, and the Room pictogram size is adjusted as Rooms are added and deleted. Windows within each Room are represented as rectangular window pictograms. From the Overview the user is reminded of the overall layout

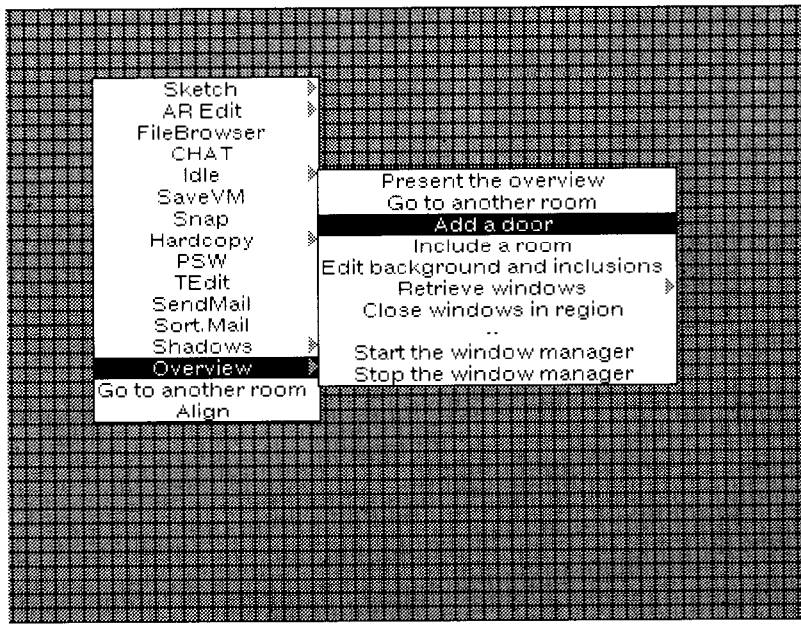


Fig. 12. Pop-up menu for invoking Rooms functions. Each of the small left triangles indicates where further expansion of the menu is possible (a standard Interlisp-D device). The menu has been expanded to show the main functions available through the menu. Since this menu is always available (by pressing the right mouse button when the cursor points to the screen background), the user is able to invoke Rooms commands even if all of the windows in a Room are deleted.

of a Room. The user can select a Room to enter by holding down the **OPEN** key while selecting a Room with the mouse. Design solution 1 (a pop-up menu) is fast, but gives only the names of the other Rooms. Design solution 2 (the Overview) takes a little longer, but gives the user much more information. The choice of one of these depends on the user's state of knowledge.

3.2.3 *Finding Windows.* Although the Room name and the shape and arrangement of window pictograms in the Overview definitely help the user's orientation, still more help is often needed to enable the user to locate particular windows or to be reminded of what particular pictograms mean.

ISSUE 9 [*Window identification*]. *How can the user identify particular windows in other Rooms from the Overview?*

DESIGN SOLUTION. *Expanding pictograms.*

The Rooms system permits the user to "expand in place" window pictograms pointed at by the mouse (Figure 13). It is worth noting that, whereas the Overview diagram is a space-multiplexed way of showing the whole view, the **EXPAND** key is a time-multiplexed technique. For reasons of speed, legibility, and versimilitude, the window is shown at full scale, as indicated by the selected Placement, instead of the information in the Room being scaled to fit the pictogram.

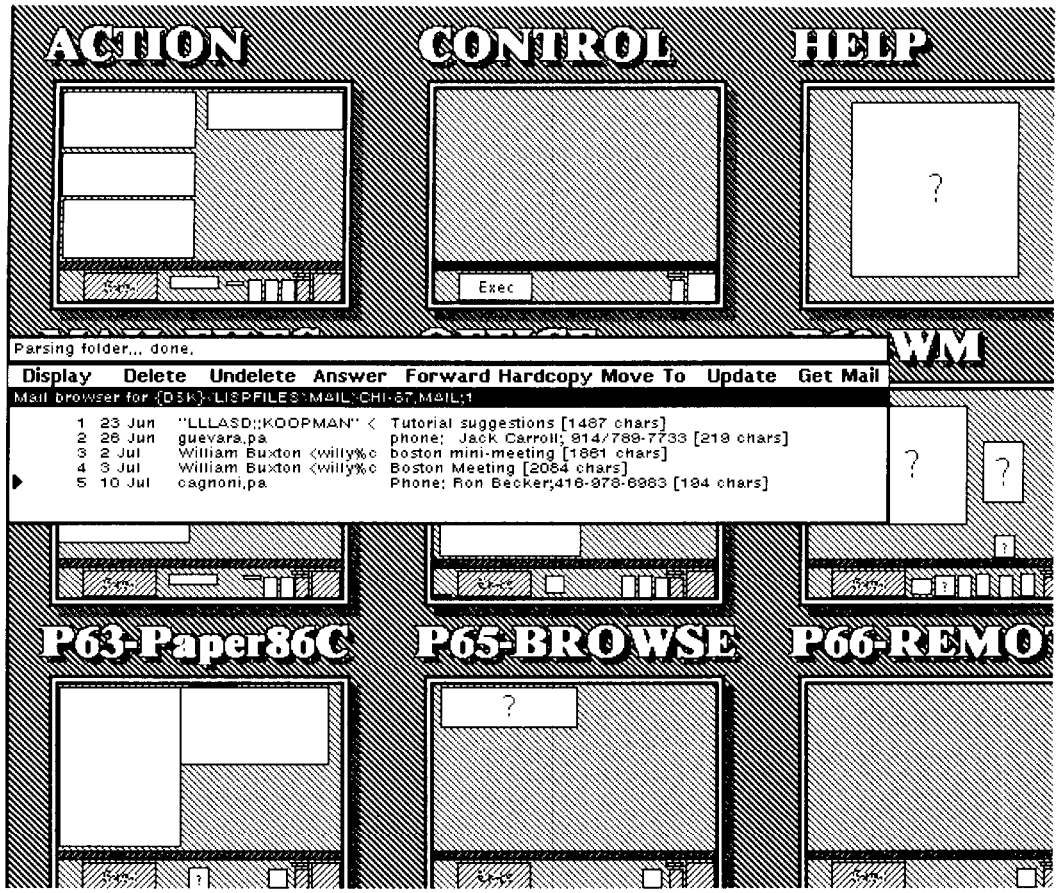


Fig. 13. Example of an **EXPANDed** window. **EXPANDing** windows is a time-multiplexed technique that allows the user to learn a large amount of information about the multiple workspaces in the system without crowding the display.

3.2.4 *Finding Which Rooms Connect.* From the Overview diagram, it is difficult to discover which Rooms have Doors to, or are included in, which other Rooms.

ISSUE 10 [*Workspace connectivity*]. *How can the user see the connections between Rooms?*

DESIGN SOLUTION. *Wiring diagrams.*

A solution to this problem is to trace out on the display a diagram showing the connections between Rooms. Figure 14 shows an example of such a diagram (**DOORS-OUT**, the set of Rooms to which the subject Room has Doors). Several such diagrams are available **DOORS-OUT**, **DOORS-IN**, **INCLUDES-ROOMS**, and **INCLUDED-IN-ROOMS**. Because of the complexity of the possible connections between Rooms and the desire not to rearrange the Overview display to simplify the connection lines (which would drastically decrease Room pictogram size), having the user interrogate one Room at a time is more successful

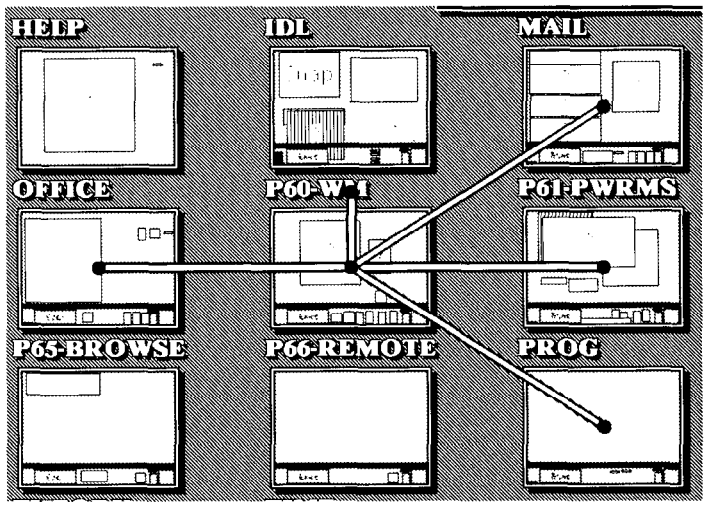


Fig. 14. Wiring diagrams. This is another time-multiplexed technique that allows the user to see which windows are connected to which others.

than asking for all the connections simultaneously. This is another case in which we fall back to the time-multiplexing of information, since showing all connections at the same time reduces the display to a tangle of lines.

3.3 Interface Issues of User Presentation Tailorability

User's workspaces change continually. Provision must therefore be made for users to reconfigure their workspaces easily: altering windows; adding and deleting Doors; creating, deleting, and renaming Rooms. All these can be expected to occur in the course of normal work.

3.3.1 *Manipulating Rooms, Windows, and Doors*

ISSUE 11 [*Room redecoration*]. *How can users manage the creation and deletion of Rooms, windows, Doors, etc?*

DESIGN SOLUTION 1. *Maintain a normal Interlisp-D environment.*

The Rooms system is designed so that users have the illusion that they are in a normal Interlisp-D window environment. Thus they can engage in all the normal Interlisp-D window manipulations: creating, destroying, copying, and moving windows or shrinking them into icons. Closing a window that exists in more than one Room brings up a menu giving the user the choice of deleting only this Placement or of deleting all Placements and closing the window itself.

DESIGN SOLUTION 2. *Persistence of window modifications.*

A related part of the design solution is that small changes users make in the course of their work persist over entering and leaving Rooms. When reentering a workspace, the user finds it arranged just as it was when he or she left it (the contents of shared windows may well have changed, of course). Modifications to a Room are accurately reflected in the Overview.

DESIGN SOLUTION 3. *Pop-up menu.*

Here, as elsewhere in the system, we maintain the principle that a basic set of system capabilities (creating Doors, going to other Rooms, going to the Overview, recovering lost windows, etc.) (see Figure 12) is maintained on a pop-up menu that is always available. One reason for this principle is to protect the user: Since completely free to design the workspace, the user could delete all the Doors from this workspace, including the Door to the Overview. Or the user could create a Door to a new Room, then enter it; this would leave him or her in a completely blank Room. In such circumstances the pop-up menu provides the user with adequate rescue controls. It does so without violating another principle, that the user should be free to determine the total physical appearance of a Room. Doors are thus accelerators that trade screen space for faster speed. In fact, the Overview just continues further along this trade-off, trading the entire screen space for rapid manipulation. This trade-off among space, speed, and robustness is the basic reason for having more than one solution to design issues.

DESIGN SOLUTION 4. *Overview commands.*

Some operations by users involve more than one Room, for example, moving windows from one Room to another or copying a Room. To make these easier, the Rooms system provides a set of commands available in the Overview (Table II). Generic commands (**COPY** and **DELETE**) can apply either to a Placement of a window in a Room or to a Room itself, according to which button on the mouse is used to select the object. Other commands (**MOVE, RESHAPE, RENAME**) apply only to one or the other.

An easy way to create a new Room, with a layout and Placements the user likes, is to press **COPY** and then select an existing Room. The system asks for the name of the new Room, then creates the new Room, and rearranges the Overview to show it (reducing the size of Room pictograms if necessary). The user could then delete any unwanted windows in the new Room by holding down **DELETE** while selecting the window pictogram with the mouse "Placement button" (left button). A similar mechanism can be used to include one Room in another.

3.3.2 *Extended Behavior and Appearance.* Although Rooms provides a number of single methods by which users can tailor their workspaces, we believe it is prudent to provide for a system's natural evolution by supplying escape hatches that enable more sophisticated and daring users to extend the system or modify it to serve their own purposes. Rooms descriptors are the mechanisms by which the advanced users in the community can achieve new effects and extensions quickly without rebuilding the system or understanding all its parts. Successful features are then given more general user interfaces.

ISSUE 12 [*Unanticipated modifications*]. *How can we provide a means for systems programmers to evolve the system by creating more complex effects?*

DESIGN SOLUTION. *Editing of Room descriptors.*

Each Room can have associated with it expressions that will be evaluated in conjunction with certain significant events (creating a Room, leaving a Room,

Table II. Overview Commands

Command	Mode key(s)	Description
Overview commands for manipulating Placements[a]		
Move	**MOVE, M**	A Placement is moved within a Room or from one Room to another Room.
Shape	**SHAPE, S**	A Placement is reshaped within a Room or into another Room.
Copy	**COPY, C**	A copy of a Placement is made in another Room.
Delete	**DELETE, D**	A Placement is deleted from a Room.
Expand	**EXPAND, ?**	The window associated with a Placement can be temporarily viewed as the Placement indicates.
Overview commands for manipulating Rooms[b]		
Enter	**OPEN, O**	The Overview is left and the indicated Room entered.
New	**NEW, N**	A name is requested and the Room is renamed.
Edit	**EDIT, E**	A structural description of the Room is made available for editing. The changes take effect when the editing is finished. More than one Room may be modified at a time, permitting copying structure from one description to another.
Copy	**COPY, C**	A name is requested and a copy of the Room is made.
Rename	**RENAME, R**	A name is requested and the Room is renamed.
Delete	**DELETE,D**	The Room is deleted.
Doors-out	**DOORS-OUT**	The set of Rooms to which Doors in the indicated Room lead is displayed in Figure 14.
Doors-in	**DOORS-IN**	Like Doors-out, but the set of Rooms that have doors into the indicated Room is displayed.
Includes	**INCLUDES**	The set of Rooms that the indicated Room includes is displayed in a diagram similar to Figure 14.
Included-in	**INCLUDED-IN**	Like Includes, but the set of Rooms in which the indicated Room is included is displayed.
Overview commands for manipulating collections of Rooms[c]		
Save	**SAVE**	A set of Rooms is indicated by selecting maps (default is all the Rooms), a file name is requested, and a description of the set of Rooms is written onto the file.
Restore (Augment)	**RESTORE (AUGMENT)**	A file name is requested, and a set of Rooms is reconstituted from the descriptions on that file. The set of current Rooms is replaced (extended) with this reconstituted set.

[a] These commands are issued by depressing a mode key and buttoning the pictogram for the Placement with the left button on the mouse.

[b] These commands are issued by depressing a mode key and buttoning the map for the Room with the right button on the mouse.

[c] These commands are issued by selecting button-shaped windows appropriately labeled.

placing a window, hiding a window, saving a Room on a file, or restoring a Room). These are made available to the (advanced) user by making a descriptor of the editable Room through the normal structured program editor. A description of the background for the Room, an expression in a layout language (Table III), is

Table III. Layout Language for Rooms Background Graphics

Specification	Description
(WHOLEBACKGROUND shade)	Shades the whole background.
(WHOLEBACKGROUND bit map)	Tessellates the whole background with the bit map.
(BOX shade region operation)	Shades a region using the graphic operation. Graphics operations are replace, paint, erase, and invert.
(FRAME shade region width operation)	Frames a region with a shaded frame of a particular width using the graphic operation.
(BITMAP bit-map region operation)	Places the bit map clipped by the region using the graphic operation.
(TESSELLATE bit-map region operation)	Tessellates the region with the bit map using the graphic operation.
(TEXT string font position operation)	Places the text in the font starting at the position using the graphic operation. In this operation graphics operations include an extension for describing drop shadows and smearing.
(BORDER shade)	Sets the border region (from the edge of the screen to the bezel of the display) to be the shade.
(IF (condition spec . . . spec) . . .)	Carries out the specifications contained in the first clause whose condition is satisfied. Conditions are Interlisp-D forms treated as predicates.
(EVAL action)	Escape to Interlisp-D: Action is an Interlisp-D form that is evaluated, presumably for its graphic effect on the background.
(COMMENT . . .)	A message to humans that has no effect on the background graphics.

Notes:

The background graphics for a Room is described by a list of graphic specifications that are executed in order, each affecting the results of the ones carried out before it.

All arguments can be either literal (for simple expression of the common cases) or forms to be evaluated (another escape clause to Interlisp-D).

also part of the Room descriptor. By holding down the **EDIT** key while pointing to a Room in the Overview, the user can "turn the pictogram of the Room around to reveal the clockwork mechanisms on the back" (Figure 15). On completion of editing, the system checks the structure of the Room descriptor to provide error protection before rendering the Room. This editing facility has been used to build elaborate graphical backgrounds and for other tasks, such as checking whether certain files are loaded before entering a Room.

3.3.3 *Saving and Restoring.* Finally, the system will not be successful unless it is possible to save, restore, and add to a user's suite of workspaces via information stored on files. If a system crash or reload/reinitialization means that the user must rebuild a suite of Rooms from scratch, few users will persist, and Rooms will not be successful in helping users to manage their screen space.

ISSUE 13 [*Saving/restoring workspaces*]. *How can a user save and restore workspace?*

DESIGN SOLUTION. *Save/restore buttons and Room descriptions.*

It should first be realized that a Room cannot be saved directly. Rooms contain complex structures including windows, large bit maps, file pointers, network

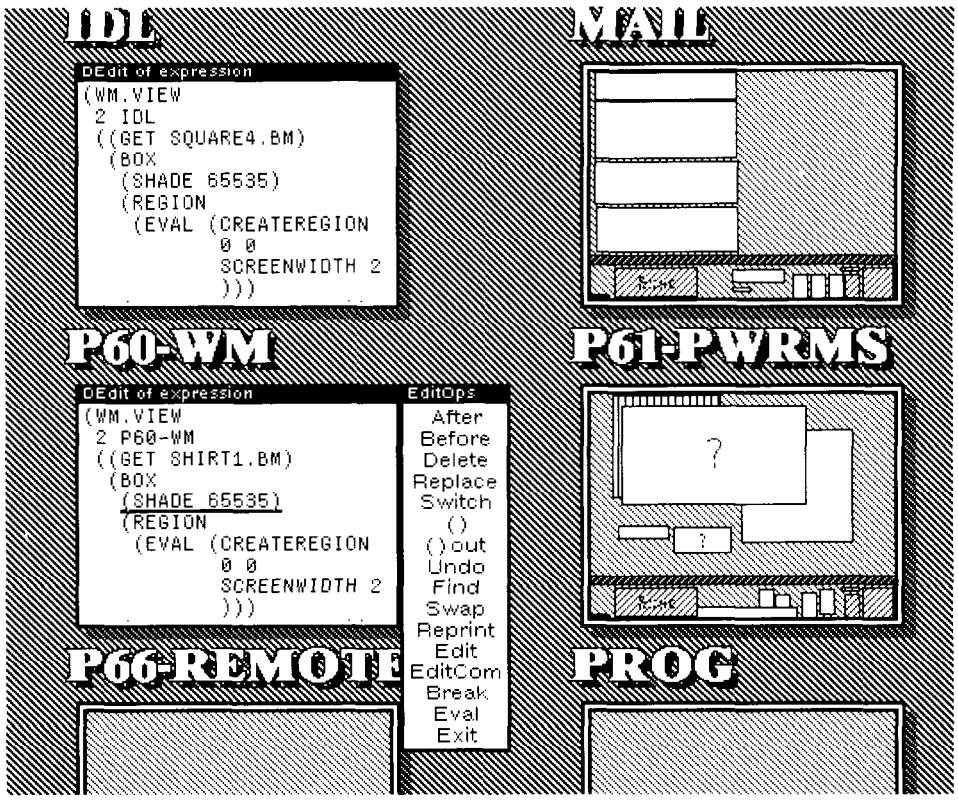


Fig. 15. Editing the Rooms description by "turning the window around to get at the clockwork on the back." Alterations to a Room will take effect as soon as the user exits from the editor for that Room.

streams, and other objects difficult to save. For this reason it is necessary to create an abstracted description of each Room such that the Room could be largely reconstructed from the description. Although the issues here are those farthest removed from the central mission of the Rooms system, they are also the most complex to program. Entries can be provided to users' initialization profiles so that, even when the user starts up a completely new system, the same Rooms structure will be created, complete with text editors open to the same files, etc. For the few cases in which this is not possible (e.g., an application not registered with the system), dummy windows still appear with indications of original titles to aid the user in remembering where he or she was. A facility is provided to allow users to save or load selected Rooms as a mechanism to enable them to design and exchange window designs and applications.

4. DISCUSSION

In Rooms, we adopted the multiple-virtual-workspace solution to the small-screen problem. This solution is used in conjunction with other techniques: Windows can be opened and closed, shrunk and overlapped, even moved

off-screen within the same virtual workspace. In addition to the analysis presented above, our use of the multiple-virtual-workspace solution also reflects our experience with an earlier window manager prototype, called BigScreen (Figure 16), in which we explored the large single-virtual-workspace technique. We observed that windows laid out in this space tended to cluster into those necessary to carry out particular tasks, and that user movement quickly reduced to simple jumps into easily named areas on the plane (e.g., MAIL). To put it another way, the windows tended to get organized around tasks (generic tasks like mail reading or specific projects), and the user mainly just wanted to switch among familiar tasks. Task switching seemed to have a nonspatial representation in the user's mind: Tasks were easy to name ("read the mail"), but hard to locate in space (Is mail north or south of here?). In fact, the relative arrangement in space of the tasks was largely irrelevant, and the geometrical constraints entailed by arranging the task windows on a two-dimensional plane were just a nuisance. We found ourselves building accelerators (both spatial overviews and nonspatial pop-up menu lists) for task switching. The conceptual step to Rooms was small, essentially dropping the single extended workspace that was a nuisance and retaining the multiple spatial contexts that worked well. It should be noted, however, that there may be applications with a very different mental structure for the user (e.g., browsing unfamiliar documentation or computer-aided design) in which either the spatial proximity or physical analog properties of a large virtual workspace could be used to advantage.

As in other systems in which not all of the information is visible at any given moment, the Rooms system faced the questions of (1) navigation and (2) simultaneous access to separated information. We now contrast the design solutions employed in Rooms with those of other systems.

Navigation. Rooms provides a pop-up menu listing the Rooms and an Overview showing pictograms of all Rooms and their Placements. This is like the Smalltalk Project Browser or, more particularly, the Cedar desktop overview. Chin's Room system, by contrast, has no such Overview. Our experience suggests that navigation tends to be easier in a multiple-virtual-workspace system than in either a large single workspace or a hypertext system. In a large single workspace, an overview picture of the workspace tends to make the details of the overview picture too small to use. In a hypertext system, the workspace is fragmented into so many pieces that either the entire structure is too large to show, or, although it can be shown, the details of the overview graphics are too small to use or the user must settle for partial browsers (e.g., the user lays out the structure for one type of link). In a multiple-virtual-workspace system like Rooms, Smalltalk Projects, Cedar desktops, or CCA, the multiple workspaces provide a level of aggregation appropriate for overview displays.

Like hypertext systems, Rooms does provide (in the Overview) for querying the connectivity of the structure. Rooms also provides a trace of the user's motions through the space via its use of Back Doors. This is similar to the dynamic stack of Chin's Room system, but it has the advantage of not requiring any additional mechanism within the system, since Back Doors are Interlisp-D windows (and so can have actions associated with input events) just like any other Door. Finally, Rooms provides Doors, parallels for which exist in many

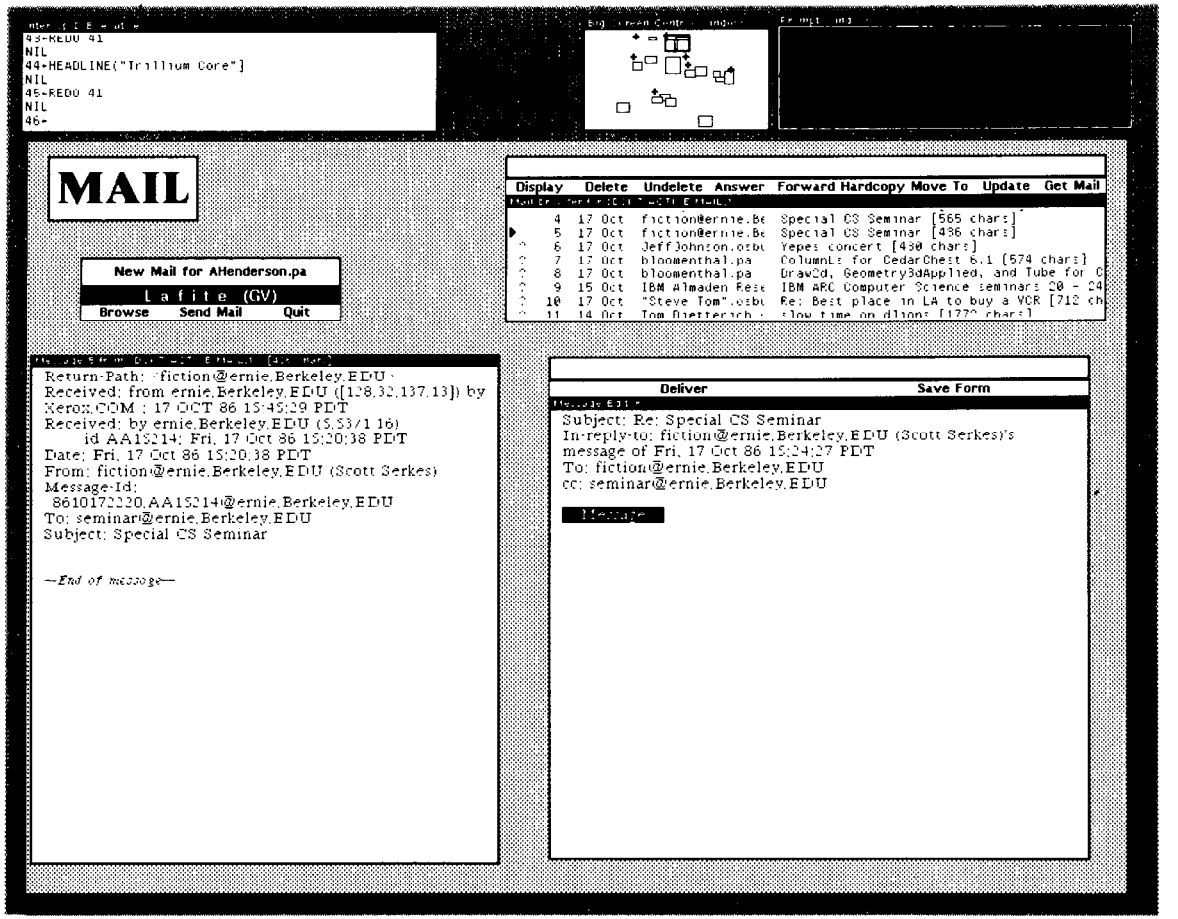


Fig. 16. BigScreen. This screen image from the BigScreen experimental system, a precursor to Rooms, shows a framed view of the mail area of a large virtual workspace. An overview of the workspace is presented in the control window (at top center). The overview contains pictograms of windows, named locations (marked with +), and the position of the current view (the heavy rectangle). The user moves the view either by selecting a named location (presaging Rooms) or by repositioning the view rectangle in the control window. Windows in the frame do not move (presaging Rooms's panels), and windows are used to identify named locations (presaging Rooms's background graphics). This system was considered unsatisfactory because of the need to have the same window in more than one location in the workspace.

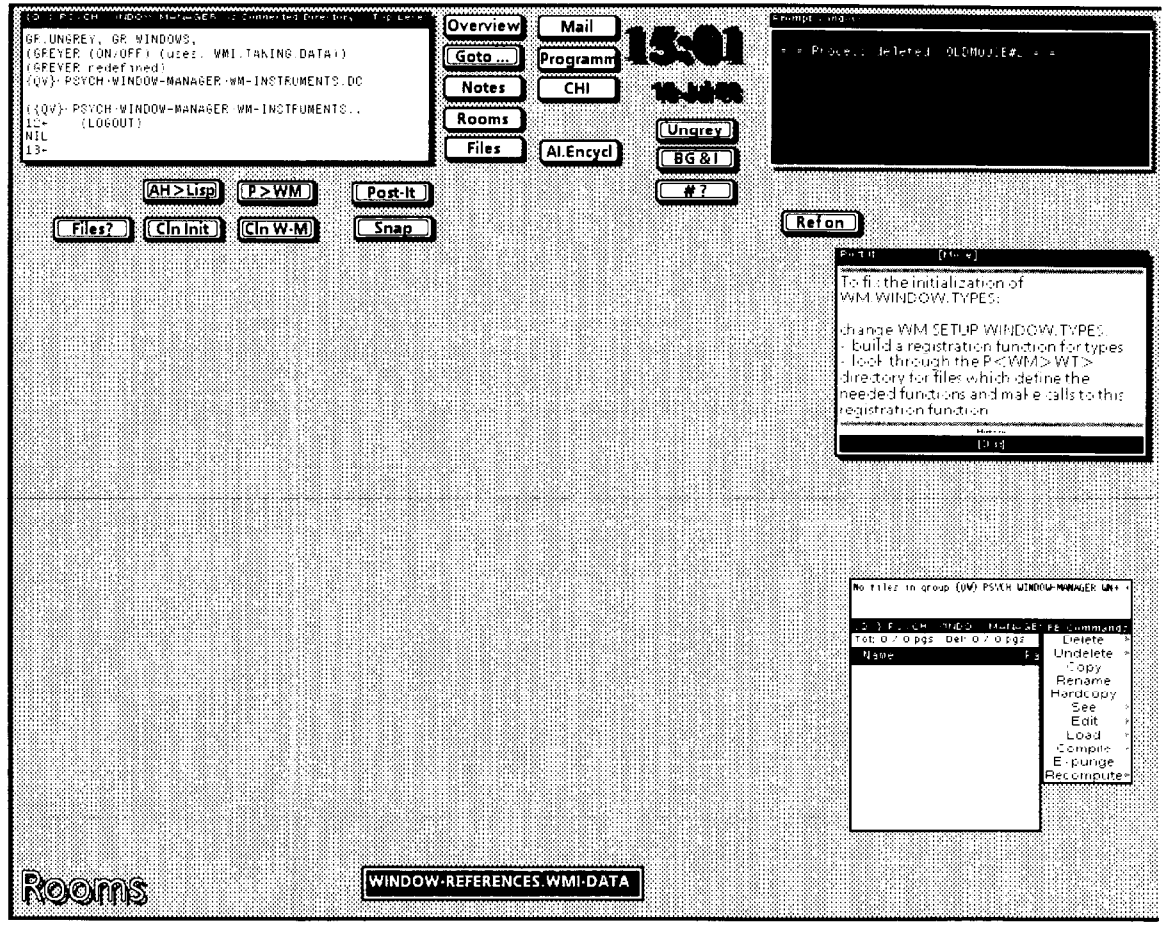


Fig. 17. Example of how the effective availability of more screen space enables a user to use the screen space for shortcuts. The buttons each call small programs for common tasks, such as connecting to a directory. This would be a lavish use of space without the Rooms system.

other systems (often even *named* "doors"). Some systems limit the functionality of these doors to motions within the structure of the system (in Smalltalk Projects and many others the doors move only up and down the hierarchy). In Rooms, doors support movement, as in hypertext systems, to any other Room and even to the Overview.

Simultaneous access to separate information. An advance of Rooms over previous systems is in the mechanisms worked out to share individual windows among workspaces through Placements and to share collections of windows through inclusion. In Smalltalk Projects, windows are partitioned among the workspaces. They cannot easily occur in several workspaces at the same time. Cedar multiple desktops does have a facility for allowing this, similar to our Placements, but the interface mechanisms that allow the user to take easy advantage of this facility are not developed. In large virtual-workspace systems, like Dataland, and in distortion systems, information can be moved among work areas, but only at the expense of destroying existing arrangements. This difficulty also appeared in our large virtual-workspace system BigScreen and was a factor in our progressing to the Rooms design.

Our early use of the Rooms system suggests the following ways in which it seems to impact the use of screen space: (1) A greater amount of information is kept in the total workspace (e.g., more windows and larger windows); (2) screens are less crowded (because information is distributed among workspaces, each related to a single task); and (3) users find new ways of consuming screen space for their convenience, particularly by using accelerators for common tasks.

With the pressure for screen space reduced, we have discovered a tendency to use some of this extra space to reduce the time required to do common tasks. Figure 9a shows one such use: a Room with several mail browsers already laid out. Normally, the user would reduce these browsers to icons or close them altogether. But, with a special mail Room, the browsers can be left open, ready for instant use, both saving considerable time and allowing the user to have a better overall picture of the incoming mail. Furthermore, because the layout will not be disturbed by the next task, the user can afford to spend more time carefully arranging the windows in the Room for maximum productivity. Another example is Figure 17. Here the user has created special "buttons" (icons that execute arbitrary code when selected) for a number of tasks. The buttons, which can be created in seconds, function like macrooperators and seem to boost user efficiency substantially.

5. CONCLUSION

We would argue that a major purpose of research into human–computer interaction is to discover and analyze *key constraints* that are the drivers of human performance and to use the representation of the problem gleaned from that analysis as tools for thought in design.

In the present case our analysis concludes that part of the "electronic messy-desk problem" derives from a screen-space resource contention. The severity of this contention depends on screen size and on the locality of window reference for the activities in which the user is engaged. We have attempted to use the

representation of the problem provided by this analysis as the basis of a design for a virtual-workspace system Rooms. In the course of a design there arise a number of issues that must be faced in order to maintain the viability of the design. In Rooms these arise from navigation among workspaces, from simultaneous access to information in different workspaces, and in tailoring Rooms for particular application and appearance.

Complementing this derivation of design from theory are new perspectives on theory from experience with implementing and using designs. In the present case the emergent needs for sharing windows and sets of windows and the support needed for navigation have elaborated the structure of the phase-and-transition window faulting model. These insights, if not simply artifacts of the particular design, can offer new grist for our theoretical mill. We believe that these two processes, theory to design, design to theory, must complement one another for good design—or good theory.

ACKNOWLEDGMENTS

The resource-contention theory on which the design of Rooms is based derives from work done jointly with Misha Pavel and Joyce Farrell. The authors would like to thank Melissa Monty for discussions on the relevance of task switching to workspaces, John Maxwell and Dan Swinehart for discussions on Cedar windows, and Sue Booker for advice on graphics.

REFERENCES

1. BOLT, R. A. *The Human Interface.* Lifetime Learning Publications, Belmont, Calif., 1984.
2. CARD, S. K., AND HENDERSON, D. A., JR. A multiple virtual-workspace interface to support user task switching. In *CHI '87 Conference on Human Factors in Computing Systems* (Toronto, Canada, Apr. 6–9). ACM/SIGCHI, New York, 1987.
3. CARD, S. K., PAVEL, M., AND FARRELL, J. Window-based computer dialogues. In *Human–Computer Interaction—Interact '84*, B. Shackel, Ed. North-Holland, Amsterdam, 1985, pp. 239–243.
4. CHAN, P. P. Learning considerations in user interface design: The Room model. Tech. Rep. CS-84-16, Dept. of Computer Science, Univ. of Waterloo, Ontario, Canada, 1984.
5. DENNING, P. J. The working set model for program behavior. *Commun. ACM 11*, 5 (May 1968), 323–333.
6. DENNING, P. J. Virtual memory. *ACM Comput. Surv. 2*, 3 (Sept. 1970), 153–189.
7. DENNING, P. J. Working sets past and present. *IEEE Trans. Softw. Eng. SE-6*, 1 (Jan. 1980), 66–84.
8. DISESSA, A. A principled design for an integrated computational environment. *Hum.–Comput. Interaction 1*, 1 (Jan. 1985), 1–47.
9. DONAHUE, J., AND WIDOM, J. Whiteboards: A graphical database tool. *ACM Trans. Off. Inf. Syst. 4*, 1 (Jan. 1986), 24–41.
10. ENGELBART, D. C., AND ENGLISH, W. K. A research center for augmenting human intellect. In *Proceedings of the AFIPS Fall Joint Computer Conference*, vol. 33 (San Francisco, Calif., Dec. 9–11). AFIPS Press, Reston, Va., 1968, pp. 395–410.
11. FEINER, S., NAGY, S., AND VAN DAM, A. An experimental system for creating and presenting interactive graphical documents. *ACM Trans. Graph. 1*, 1 (Jan. 1982), 59–77.
12. FISHER, S. S., MCGREEVY, M., HUMPHRIES, J., AND ROBINETT, W. Virtual environment display system. In *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, F. Crow and S. M. Pizer, Eds. (Chapel Hill, N.C., Oct.). ACM, New York, 1986. To be published.
13. FURNESS, G. Generalized fisheye views. In *CHI '86 Conference on Human Factors in Computing Systems*, M. Mantei and P. Orbeton, Eds. (Boston, Mass., Apr. 14–18). ACM/SIGCHI, New York, 1986, pp. 16–23.

14. GOLDBERG, A. *Smalltalk-80: The Interactive Programming Environment.* Addison-Wesley, Reading, Mass., 1984.
15. HALASZ, F., MORAN, T., AND TRIGG, R. NoteCards in a nutshell. In *CHI '87 Conference on Human Factors in Computing Systems* (Toronto, Canada, Apr. 6–9). ACM/SIGCHI, New York, 1987.
16. HEROT, C. F. Spatial management of data. *ACM Trans. Database Syst. 5,* 4 (Dec. 1980), 493–514.
17. HURST, J., AND WALKER, K., EDS. *The Problem-Oriented System.* MEDCOM Press, New York, 1972.
18. KAHN, K. C. Program behavior and load dependent system performance. Ph.D. dissertation, Dept. of Computer Science, Purdue Univ., West Lafayette, Ind., Aug. 1976.
19. MADISON, A. W. *Characteristics of Program Localities.* University Microfilms International, Ann Arbor, Mich., 1982.
20. MCGREGOR, S. The viewer window package. In *The Cedar System: An Anthology of Documentation,* J. H. Horning, Ed. Tech. Rep. CSL-83-14, Xerox Palo Alto Research Center, Palo Alto, Calif., 1983.
21. MONTY, L. In *Human–Computer Interaction—Interact '84,* B. Shackel, Ed. North-Holland, Amsterdam, 1985, pp. 603–609.
22. ROBERTSON, G., NEWELL, A., AND RAMAKRISHNA, K. The ZOG approach to man–machine communication. *Int. J. Man–Machine Studies 14,* 4 (May 1981), 461–488.
23. SMITH, D. Pygmalion. Ph.D. dissertation, Dept. of Computer Science, Stanford Univ., Stanford, Calif., 1975.
24. SMITH, D. C., IRBY, C., KIMBALL, R., VERPLANK, W., AND HARSLEM, E. Designing the Star user interface. *Byte 7,* 4 (Apr. 1982), 242–282.
25. SPENCE, R., AND APPERLY, M. Data base navigation: An office environment for the professional. *Behav. Inf. Technol. 1,* 1 (Jan. 1982), 43–54.
26. SUTHERLAND, I. E. Sketchpad: A man–machine graphical communication system. In *AFIPS Spring Joint Computer Conference,* vol. 23, 1963, pp. 329–346.

Received July 1986; revised November 1986; accepted November 1986