

A Multiple, Virtual-Workspace Interface to Support User Task Switching

S. K. Card and D. A. Henderson, Jr.

UIR-R-1987-02

A Multiple, Virtual-Workspace Interface to Support User Task Switching

Stuart K. Card and Austin Henderson, Jr.

Intelligent Systems Laboratory
Xerox Palo Alto Research Center
Palo Alto, California 94304

Abstract

An interface is presented that is designed to help users switch among tasks on which they are concurrently working. Nine desirable properties for such an interface are derived. It is argued that a key constraint to building interfaces that support task switching is that low user-overhead switching among tasks requires a large amount of display space, whereas actual display space is limited. A virtual workspace design is presented that greatly speeds the inevitable task-switching induced window faulting. The resulting interface is presented as a study in theory-based human-interface design. It is shown how in this case theory is important in inspiring a design, but design entailments outside the theory raise new issues that must be faced to make the design viable. These design experiences, in turn, help inspire new theory.

Ce papier décrit un interface conçu pour aider les utilisateurs à choisir parmi des tâches auxquelles ils travaillent en parallèle. Neuf particularités souhaitables pour un tel interface sont déduites. Une contrainte majeure concernant la productivité des utilisateurs est que changer de tâche sans beaucoup d'effort par l'utilisateur nécessite une grande surface d'écran, alors que la surface disponible est limitée. L'on présente ici une conception d'espace virtuel de travail qui accélère notablement les fautes de fenêtres inévitablement causées par le changement de tâche. L'on montre comment les conséquences de la conception de base soulèvent d'autres problèmes qu'il faut résoudre pour rendre viable la conception finale.

Introduction

Most user interfaces are designed to help the user perform particular task to completion. But users actually switch back and forth among several concurrent tasks [1]. Without special interface support, task switching can lead to major difficulties. In traditional command-oriented systems, the user is usually able to see information from only one of the tasks at a time. As a

This work was supported in part by NASA AMES Grant NAG 2-269.

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.

consequence, the state of tasks not on the screen is hard to remember and the user may be forced to extreme adaptations such as writing information from one task on paper, then typing it into to another task. On the other hand, window-oriented systems allow the user to see information from several tasks, but severe conflicts among tasks for the use of screen space may lead to high overheads as users must move, reshape, or scroll windows or shrink and expand icons.

In this paper we analyze the problem posed by task switching and propose a user interface. The interface appears to satisfy a number of properties desired for supporting task switching.

Interface Properties to Support Task-Switching

Bannon et al [2] identified a number of reasons why users switch from one task to another: (1) digressing to do tasks that users are reminded of while performing another task ("While I'm At It tasks"), (2) timesharing among concurrent demands, (3) tasks with long waits, (4) subtasks, and (5) "snags" such as running out of file space. To these reasons, we might add others such as (6) interruptions from outside ("please write your section by 5 pm"), (7) shifting to another current project (because it's scheduled now or because some relevant mail came in), and (8) shifting to a specialized environment (say, to draw a figure). There are probably many more. The point is that interruptions and other sorts of task switching are an important aspect of user activity. Indeed, studies have shown that the average time a manager works on a single desk top is only about 15 min [12]. Task switching occurs for activities measured over minutes (such as those described in Bannon et al), where task switching time and resumption are especially important, and it occurs for activity measured over days, where the memorial aspects of remembering the activities, their pieces, and their state are especially important.

Bannon et al suggested six issues that an interface to support task switching should engage: (1) reducing mental load when switching tasks, (2) suspending and resuming activities, (3) maintaining records of activities, (4) functional grouping of activities, (5) multiple perspectives on the work environment, (6) interdependencies among items in different workspaces. Before proposing an interface which, we believe, satisfies most of these properties, we first proceed to suggest a refinement of Bannon et al's list.

Task Switching Properties

There are essentially two problems associated with task switching per se: the amount of time it takes and the mental

complexity of remembering how to invoke the other task and of trying to get into mental context. Task switching can be time-consuming because the user must first put away the current task, then get out the tools for the second task. On a computer system this could involve saving files, looking through directories to find the names of other files, loading a program, and pausing while the user tries to remember a file name or a program name or consults a notebook. So whatever else it is, we would wish an interface to have the property:

A1. Fast task switching.

And since many task switches are short subtasks or digressions, we would want also wish the companion property:

A2. Fast task resumption.

But the most important property we would wish relates to the second problem of task switching: the mental complexity of remembering where the user was in a resumed task. Not only may the user consume time recalling his previous mental state, file names, programs used, etc, he might actually never be able to resume the same path. One thinks of lost ideas for algorithms, lost verses to poems, projects pursued differently because the pursuer was interrupted and forgot the details of what he was doing. So our third property is:

A3. Easy to re-acquire mental task context.

Information Access Properties

Let us examine further the problem of mental complexity. Almost any knowledge-intensive task is complex and requires too much memory for a person to do efficiently in his head. Human working memory is severely limited. The general solution is to use the environment as an auxiliary to the head--to use notes, markers, diagrams, or the arrangements of piles of notes as a form of external memory linked to and augmenting the internal memory inside the user's head. The question is how to use the environment, in particular how to use the computer display, to support user multiple task activity?

Like any memory, an external memory can be characterized by capacity and access time. Consider first capacity. Imagine a user doing some particular task such as writing a paper using a window-oriented interface (to be more concrete, we will assume the Interlisp-D user environment). To do this Task, he uses on his electronic "desk-top" several specialized window-oriented objects we may call Tools: a text-editor, a file-browser, a prompt window, a clock (See Fig. 1). Some Tools may occur in more than one instantiation: one text-editor window containing the main text, another text-editor window containing the references, a third text-editor window containing a table from the paper. We call each of these an Engaged Tool to refer to the combination of Tool and contained data that makes it unique.

Each of these Engaged Tools consumes space on the display screen of our system. If each Task has several Engaged Tools, and if the user is to switch back and forth among multiple tasks, then there will be a substantial number of Engaged Tools to which the user needs access. More particularly, there will be a large amount of information contained within these Engaged-Tools to which the user needs access to do his task. So we have as another required property:

B1. Access to a large amount of information.

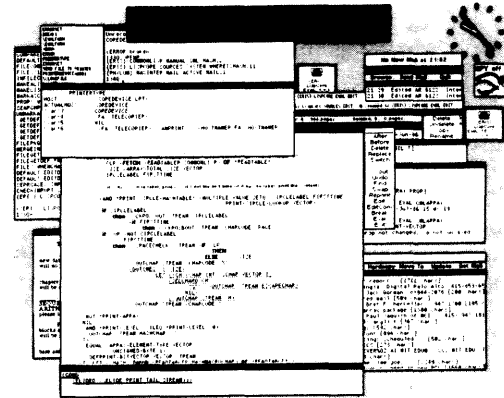


Fig. 1. Overlapped window display. Since the workspace is necessarily small, it is easily overloaded and becomes a cluttered desk, causing the user to do unnecessary work.

The companion property is

B2. Fast access to information.

The essence of the problem is that in attempting to satisfy task switching properties A1, A2, and A3 by using an external memory, property B1, large information access, and property B2, fast access contend with each other. The screen is far too small to hold all the Engaged Tools for all the Tasks at the same time (and even if it could, the user would still have some difficulty searching for what he wanted). But if there is not enough space for each Engaged Tool, then each time the user needs a tool that is not present (or is covered by some other Tool in an overlapped window, or is available only by expanding an icon), there will be a time-consuming "Tool fault" while that Tool is readied for use (by starting it from a command or a menu or by expanding an icon or by making it be the top window or by resizing or moving other windows). Thus the user can gain access to more information at the cost of overhead activities that increase access time. To emphasize the problem of overhead, we add among our list of desired properties:

B3. Low Overhead

Phases and Transitions

The essential insight for the design to be presented in this paper is that the grim tradeoff between the amount of information available and access time can be broken by taking advantage of the dynamic characteristics of user activity and information access. Even though users switch among tasks, they are actually engaged in only a single task at a time. Studies of memory access by computer programs [6] show that programs pass through a series of "phases", with "transitions" between the phases. In each phase the program accesses repeatedly some cluster of (not necessarily distinct) memory locations. In the next phase, it accesses another cluster of locations. In one study [10], programs spent 98% of their time within phases and 2% in transitions, yet 40-50% of the page faults occurred during the transitions. Preliminary results shows this clustering of activity also occurs in user interaction whether measured by inter-reference interval [4] or bounded locality interval [9]. Fig.

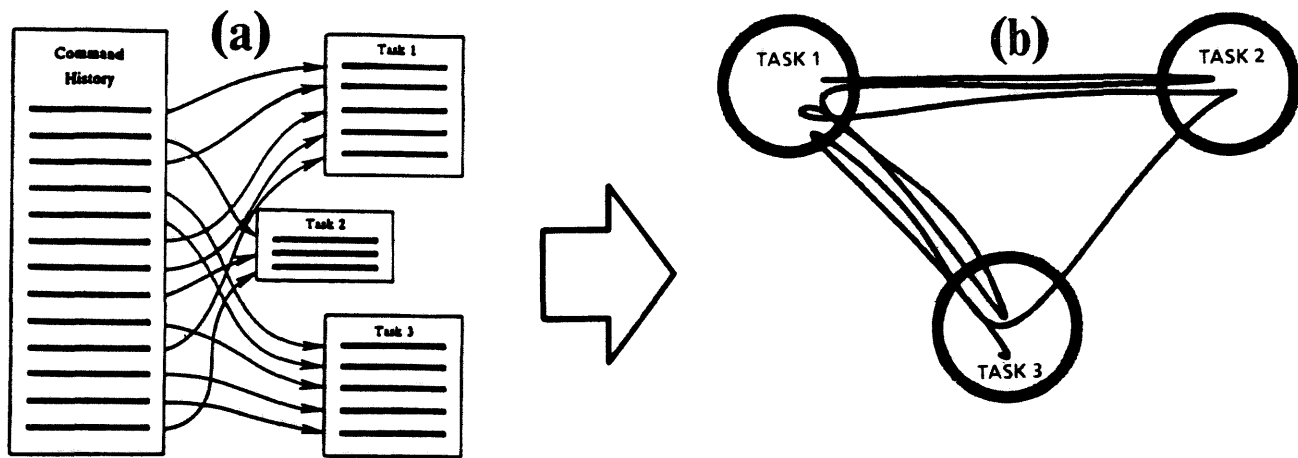


Fig. 2. Representation of user activity. (a) As represented by Bannon et al [1]. (b) In terms of phases and transitions.

2a shows Bannon et al's representation of user activity [1]. Fig. 2b shows this same activity represented in phase and transition form. These considerations suggest we can break the tradeoff between size and access time if we set up independent workspaces around each Task and allow fast transitions among them.

Interactions Among Tasks

One more issue remains in our analysis. Tasks are not necessarily independent of each other. Two Tasks may wish to share the same Engaged Tool (e.g., the same prompt window or the same to-do list). Therefore we need the property,

C1. Engaged Tools sharable among several Tasks.

Furthermore, Tools such as an alarm clock or the system prompt and typin windows may be useful to have in most Rooms. We thus need,

C2. Collections of Engaged-Tools sharable among tasks.

And finally, the same Engaged Tool may play a different role in different Tasks and there may be a different amount of space for it. Shared Tools need to be adaptable to their various Task environments. So we need,

C3. Task-specific presentations of shared Engaged-Tools.

This then is the set of properties we wish an interface that supports task-switching. We now turn our attention to describing a design to meet these goals.

The Rooms Design

From our analysis, we expect user activity to be divided into phases. We expect further that a large number of window faults will occur between major phases. We therefore arrange things so that phase changes can be accomplished in single rapid action by the user, drastically reducing the cost of major phase transitions.

Multiple Virtual Workspaces

To make these rapid transitions possible, we provide the user with a number of screen-sized workspaces called Rooms.

Fig. 3 shows two typical Rooms. In each Room, there are a number of small icon-like objects called Doors. When a Door is selected with the mouse, the user has the illusion of transiting to a new Room, containing other windows. Fig. 4 shows the basic functional structure of Rooms. Each Room is related to a different major Task, such as reading the mail or working on a particular project. In the Room are a number of Engaged Tools related to the task.

The basic notion of the Rooms scheme is therefore simple. But before this basic notion can be successful, a number of issues entailed by the basic notion arise, each of which must receive a solution. The design solution to these issues, which may have little relation to the main problem Rooms is designed to solve, further develop the design. These design solutions may precipitate other design issues. The design is viable when no fatal issues remain unresolved. The list of design issues and their design response are summarized in Table 1. They can be grouped under the headings: Task interactions, navigation, and tailorability.

Task Interactions

Tasks can interact by sharing Engaged Tools. The first three interactions have the same solution and are best treated together:

ISSUE 1. Multiple instances of Engaged Tools. It is obvious that some Engaged Tools, such as the executive window where the user can type commands, need to be able to appear in more than one place. But a window, by definition, only has a single location.

ISSUE 2. Workspace-dependent Engaged Tool locations. Tools need to be in different locations in different Rooms. Otherwise the arrangements of Tools in one Room imposes severe constraints on the locations of Tools in another Room.

ISSUE 3. Workspace-dependent Engaged Tool presentation. It may be desirable to have shared a text-editor window large in one Room, but small in another. Or it may be convenient to have the text-editor window squarish in one Room, but tall and thin in another so as to fit into a differently-arranged space. Or we may want a window to have drop shadows to emphasize it in one Room, but not in another.

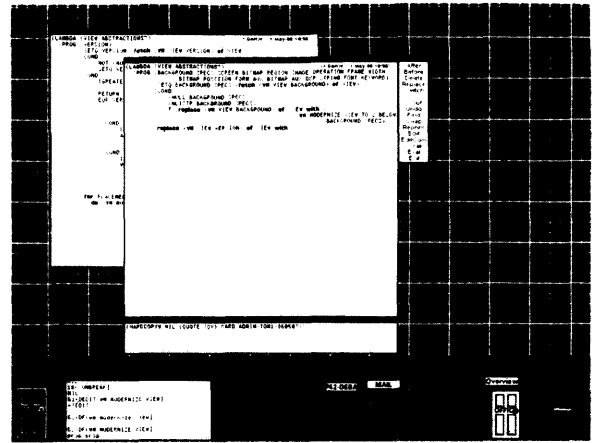
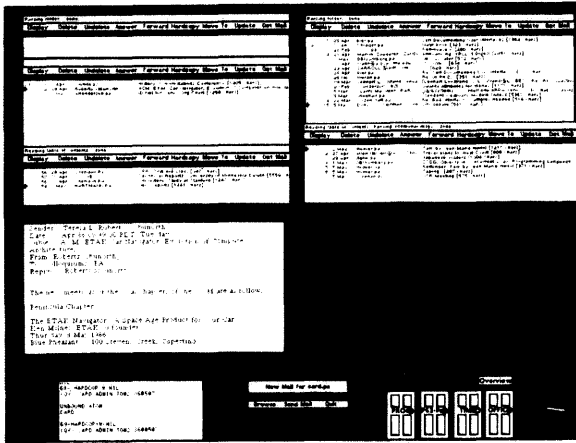


Fig. 3. Two examples of Rooms (a) A Room used for reading mail. (b) A Room used for programming. Note that both Rooms use a common control panel implemented as an included Room. Those Tools that are different in the control panel actually belong to the particular Room. In this way a single control panel is adapted for different Rooms.

In each of these cases there is a desire to have versions of the same window appearing in more than one Room and with a location, shape, and presentation that is particular to the Room. This forces us to the abstraction of a Placement (Fig. 5). A Placement is a reference to a window together with location and presentation information:

$$\text{Placement} = \text{ReferenceToWindow} + \text{LocationInRoom} + \text{PresentationAttributes}$$

A Placement divides the concept of a window, separating the tools aspect of it (the fact that it delivers certain functionality) from its appearance on the screen. Using the concept of a Placement, we can have the same window appear in different Rooms (each would have a different Placement but would refer to the same Window), we can have the locations and shape of the window be different in the different Rooms, and we can even have presentational aspects, such as whether the window has drop shadows, be different in different rooms.

The next set of issue moves from interaction between tasks at the level of individual Tools to interaction at the level of collections of Tools.

ISSUE 4. Collections of Engaged Tools. Some groups of Tools need to be defined as a collection whose location and positional attributes remain constant across workspaces. Changes to any of the Engaged Tools in the collection need to be propagated across all the workspaces containing them. An example would be a control panel with an executive

window, a prompt window, a clock, and a system memory indicator.

The design solution here is Room inclusion. Room inclusion allows a Room to itself be included in another Room, meaning that all of the Placements of the included Room will be displayed just as if they had been in the Room. The the band of windows and icons common to both parts of Fig. 3 is a control panel, implemented as an included Room.

A final final set of task interaction issues involve the user's desire to carry Tools with him as he moves between Rooms.

ISSUE 5. Carrying Engaged Tools to other workspaces. The user may wish to bring Tools with him as he moves to another workspace. For example, he may wish to bring program code from one workspace over to a workspace where he is writing a paper.

ISSUE 6. Keeping Engaged Tools along. In some applications, windows need to be automatically associated with the user, regardless of the workspace. A user might automatically want the same control panel in all his workspaces. Or he might want to put a Tool somewhere to keep it with him wherever he goes (e.g. a bar graph showing available disk space).

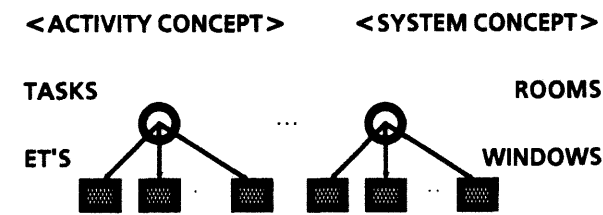


Fig. 4. Basic structure of Rooms. The labels to the left describe the design in task terms, the labels to the right in systems terms. Each major Task is associated with a Room. Engaged Tools (ET's) within each Room are seen by the user as types of windows.

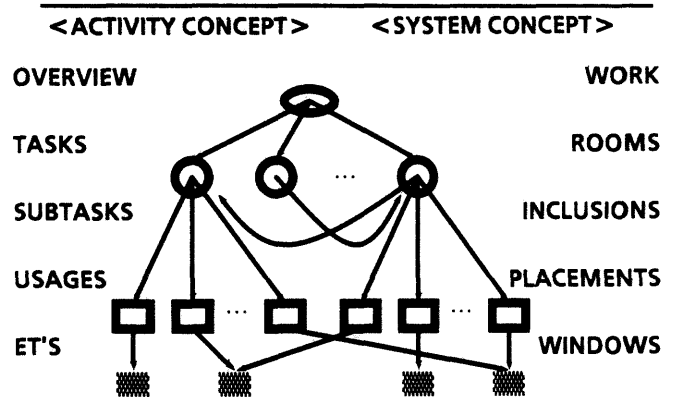


Fig. 5. Placements. In order to represent Task interactions as various sorts of Engaged Tools shared among Tasks (windows shared among Rooms), we introduce a level of abstraction called a Placement.

Our solution to the issue of carrying Engaged Tools is to give the user *Baggage* into which he packs Tools before entering a Door. He does this by using a mode key before selecting the door, putting him into a mode in which he can point to the windows he wants in his Baggage. The Baggage goes through the Door with him creating new Presentations of the windows on the other side of the Door. (The old Presentations of the windows remain). The user can also have a constant piece of Baggage called a Pocket. A Pocket is a Room dynamically included in all Rooms. Whichever windows are placed in the user's Pocket (a clock, say) will automatically occur (at the same location and with the same presentation attributes) in all Rooms.

Navigation Issues

The fragmentation of the user's workspace into a number of workspaces also creates navigational problems.

ISSUE 7. Backward workspace transitions. Users frequently want to go back to the Room they came from, but Doors are one-way. There may be no Door back and the user may not even remember the Room name.

Our design solution is to invent Back Doors. Whenever a user enters a Room, a new Door is created (in reverse video) back to the Room from which he came. It is destroyed after one use. This mechanism provides good support for interrupting and resuming Tasks.

The user still faces a serious problem of navigation, however.

ISSUE 8. User Orientation. As the number of Rooms increases, the user finds it difficult to find which Rooms exist and how to reach them. The suite of Rooms becomes an electronic maze.

Our system has two design solutions. One is a pop-up menu with the names of all the Rooms. This allows the user to get to all Rooms. The other solution is to use an Overview (Fig. 6). The main feature of the Overview is a set of Room Pictographs

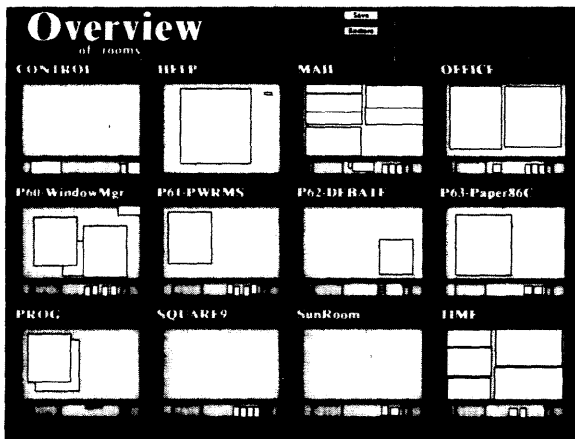


Fig. 6. Overview. The Overview contains pictograms of the Rooms arranged alphabetically. It also contains a message window for communicating with the user and buttons for saving and restoring the set of Rooms. For the user, the CONTROL window is included in every Room except the HELP Room. Windows contained in a Room because they are part of an included Room are rendered in grey. EXPANDING the window in the HELP Room provides the user with a one-page illustrated system manual.

arranged in alphabetical order. From the Overview the user is reminded of the overall layout of a Room and the Tools it contains.

Still more help is often needed, however, to enable the user to locate particular windows or to remind him what particular Window Pictographs mean or which Rooms are directly connected with which other Rooms.

ISSUE 9. Window Pictograph identification. The user needs more help in identifying or searching for particular windows.

ISSUE 10. Workspace connectivity. The user needs more help in tracing which Rooms are connected to which Rooms.

The solution to the first of these is to allow Window Pictographs to be instantly expanded one at a time, allowing the user to browse through different windows in the entire set of Rooms. The solution to the second is to have a command for drawing lines between Rooms that shows the connectivity.

Together these mechanisms solve rather thoroughly the user orientation and navigation problems. With the multiple cues of shape, size, arrangement, labels, and sequential expansions, it is possible for the user to explore easily the entire set of windows active in any of the Rooms.

User Tailorability Issues

Finally, the user's workspaces change dynamically. It must be possible for the user to add, delete, move, and reshape Tools/windows in particular Rooms quickly. The following two issues speak to this point.

ISSUE 11. Room redecoration. It must be possible to create new Rooms quickly and populate them with Tools.

ISSUE 12. Unanticipated modifications. 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 their own purpose.

In the Rooms system, several mechanisms are provided to help the user tailor his own Rooms. In the first place, simply creating, moving, deleting, and shaping windows in the usual way causes these things to exist in Rooms. Thus the Rooms simply preserve the natural interactions of the user. In the second place, special background menu entries are provided to allow the user to create new doors and other conveniences of construction. At the Overview level, it is possible to copy, move, reshape or delete window pictographs within a Room between Rooms and have the changes reflected in the Rooms themselves. And finally, we have defined a simple layout language for creating unique backgrounds for Rooms. By using an editor on this layout language, users can run arbitrary procedures on entrance and exit to a Room and can compute specialized backgrounds for Rooms. As experiments made by programming the system this way show promise, we create new abstractions and move them into the basic Rooms architecture.

Finally there is the issue of how to store a user's set of Rooms.

ISSUE 13. Saving/Restoring workspaces. The user needs to be able to save his set of workspaces, restore them, and exchange Rooms with other users.

Our design solution uses buttons for saving, restoring, and appending new Rooms in the Overview. Users can also save portions of their Room suites under different names.

Discussion

Let us now consider our basic set of desired properties in relation to the design of the Rooms system. These are summarized in Table 1.

A. Fast Task Switching

<u>PROPERTY</u>	<u>DESIGN SOLUTION</u>
<i>A1. Fast task switching</i>	--> Doors

All the tools needed for another task can be set up by a single button.

<i>A2. Fast task resumption</i>	--> Back Doors
---------------------------------	-----------------------

All the tools for an interrupted task can be resumed by a single task button.

<i>A3. Easy to re-acquire mental task context</i>	--> Rooms
---	------------------

The Room is a workspace whose window placement and content (except where affected by task interaction) is exactly as the user left it before working on another task.

B. Information Access

<i>B1. Access to a large amount of information</i>	--> Room suites
--	------------------------

The total number of windows, hence the total amount of information available in the user's entire suite of Rooms, is much larger than the user would have been able to handle on a single screen.

<i>B2. Fast access to information</i>	--> Rooms, Doors
---------------------------------------	-------------------------

The screen is kept clean of information not related to the task at hand, so more relevant information fits on the screen. The average access time to the knowledge elements is less, because there is less Tool/window faulting. It requires less time to move to another Room and access the information there than to retrieve that information from scratch.

<i>B3. Low overhead</i>	--> Rooms, Doors
-------------------------	-------------------------

Because there tend to be fewer windows per Room, there is less information faulting hence less overhead in moving and reshaping windows. Because information faulting is done en masse when switching tasks, the overhead is also less.

C. Graceful task interactions

<i>C1. Engaged Tools sharable among several Tasks</i>	--> Placements
---	-----------------------

Placements allow the same window to be in more than one workspace. Actions done on a shared window in one Room are reflected in another Room.

<i>C2. Collections of Engaged-Tools sharable among tasks</i>	--> Room inclusion, Pockets
--	------------------------------------

TABLE 1.
Desirable properties for interface and design solutions in Rooms.

A. FAST TASK SWITCHING

<i>A1. Fast task switching</i>	--> Doors
<i>A2. Fast task resumption</i>	--> Back Doors
<i>A3. Easy to re-acquire mental task context</i>	--> Rooms

B. INFORMATION ACCESS

<i>B1. Access to large amount of information</i>	--> Room Suites
<i>B2. Fast access to information</i>	--> Room, Doors
<i>B3. Low overhead</i>	--> Room, Doors

C. GRACEFUL TASK INTERACTIONS

<i>C1. Engaged Tools sharable among several Tasks</i>	--> Placements
<i>C2. Collections of Engaged-Tools sharable among Tasks</i>	--> Room inclusion, Pockets
<i>C3. Task-specific presentations of shared Engaged Tools</i>	--> Placements

Room inclusion and Pockets allow the user to build control panels of Tools that remain with him in all his Tasks.

<i>C3. Task-specific presentations of shared Engaged Tools</i>	--> Placements
--	-----------------------

The presentational aspect of Placements allows there to be windows that are shared, but which are placed, shaped, and presented independently in each Room. This prevents some undesirable interactions among Rooms.

Experience with Rooms

Early versions of Rooms have been in use since the end of January, 1986. A number of informal observations can be reported from early experience. First there is a strong psychological sense of relief that comes when the user's tasks are separated into the different Rooms. Each Room seems to have much more space with fewer windows or pixels in use on the screen. Second, users use much more total space. They have, perhaps, three times as many windows open, spread out over one to three dozen Rooms. These would occupy the area of one to two five-foot desks, in terms of raw area of screen space. Third, there seem to be three major classes of Rooms that user's make: (1) functional rooms (e.g., a mail Room), (2) project rooms (e.g., the Room for writing this paper), and (3) management Rooms (e.g., an "Atrium" for entering the system, storage Rooms, Rooms with special Tools for system initialization). Finally, because there is extra space, users make space-intensive tools (Rooms full of open mail folders, for example, or special "buttons" for doing frequent tasks). In other words they use more space to get faster working rate.

Similarities to Other Systems

Rooms continues the development of ideas begun in earlier systems. Smalltalk [8] had windows called Project Views as early as 1976. Project Views formed a tree of workspaces. Cedar [11][13] contained a fixed overview of 16 screens written by John Maxwell (no published reports). Chan [5] designed a system called Room (about which we learned only after the first public demonstration of our system, Rooms) that also, like the Smalltalk

and Cedar systems, implements multiple workspaces. The rooms in Chan's system are defined at the system level (shared by all users) and apparently do not contain windows (although they do contain activity icons and doors). Multiple rooms can be displayed at one time as bands on the screen. Sharing and interaction of tools is not addressed. Several systems have also been built in which the user is given a large virtual workspace, e.g. Dataland[3] and the Cedar Whiteboard [7]. In such single workspace systems, switching tasks can involve search. In all these systems no attempt was made to gain an analytical understanding of the task switching and screen space problems. Thus, some of the ideas in Rooms have appeared previously, but Rooms extends them and also addresses multiple occurrences of windows across workspaces and various interactions. (For a more extensive review, see [9]).

The Relationship between Analysis and Design

From our experience with the Rooms design, we have an increased appreciation of the relationship between science and design. The Rooms system began with the observation that window systems require the user to spend too much time on overhead window-manipulation tasks, especially where task-switching was involved. This led us to more formal analysis where we determined (1) desirable properties the interface should have to allow graceful task switching and (2) that a key-constraint was thrashing caused by the small screen space limitation, analogous to thrashing in virtual memory operating systems. The analysis suggested a design in which Tasks are embedded in virtual workspaces with their Engaged-Tools already laid out. This would permit the massive window faulting that was inevitable on switching tasks to occur very rapidly at the signal of a single keystroke by the user. But in order for this analysis to result in a successful system, a number of entailments of the basic design had to be successfully faced. Many of these were only marginally related to the original problem, but failure to handle any one satisfactorily could be fatal or seriously degrade the design.

Thus we can see that scientific studies of human-computer interaction may not necessarily translate automatically into successful designs. On the other hand we can see how they might serve as tools for thought, ways of structuring the problem and its key constraints in the designer's head that inspires a design not otherwise reachable from experimental programmings or intuitive design alone. Complementing this derivation of design from theory are new perspectives on theory from experience with implementing and using designs. Experience with design can also inspire a theory not otherwise reachable from other theoretical or empirical studies alone.

References

- [1] Bannon, L., Cypher, A., Greenspan, S., and Monty, M. L.. Proceedings of the ACM Human Factors in Software Conference, CHI '83 (1983), 54-57.
- [2] Bannon, L., Cypher, A., Greenspan, S., and Monty, M.. Evaluation and analysis of users' activity organization. Transcript of talk delivered at CHI '83, San Francisco, December, 1983.
- [3] Bolt, R. A. *The Human Interface*. Belmont, California: Lifetime Learning Publications, 1984.
- [4] Card, S. K., Pavel, M., and Farrell, J. Window-based computer dialogues. In B. Shackel (Ed.), *Human-Computer Interaction--Interaction '84* (London, Aug. 1984), 239-243. Amsterdam: Elsevier Science Publishers, B. V., 1985.
- [5] Chan, P. P.. Learning Considerations in User Interface Design: The Room Model (Report CS-84-16). Waterloo, Ontario, Canada: University of Waterloo Computer Science Department, 1984.
- [6] Denning, P. J. Working sets past and present. *IEEE Transactions of Software Engineering*, SE-6 (1980), 66-84.
- [7] Donahue, J. and Widom, J. Whiteboards: A Graphical Database Tool (Report CSL-84-4). Xerox Palo Alto Research Center, Palo Alto, California, June, 1985.
- [8] Goldberg, A. *Smalltalk-80*. New York: Addison-Wesley, 1983.
- [9] Henderson, A. and Card, S. Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Transactions on Graphics*, in press.
- [10] Kahn, K. C.. Program Behavior and Load Dependent System Performance. Ph.D. dissertation, Dept. of Computer Science, Purdue University, W. Lafayette, Indiana, August, 1976.
- [11] McGregor, S. The viewer window package. In J. H. Horning (Ed.). *The Cedar System: An Anthology of Documentation* (Report CSL-83-14). Palo Alto, California: Xerox Palo Alto Research Center, 1983.
- [12] Mintzberg, H.. *The Nature of Managerial Work*. New York: Harper and Row, 1973.
- [13] Teitelman, W.. Ten years of window system - A retrospective view. In Hopgood, F. R. A., Duce, D. A., Fielding, E. V. C., Robinson, K., Williams, A. S. (Eds.). *Methodology of Window Management*. Berlin: Springer-Verlag, 1986.

