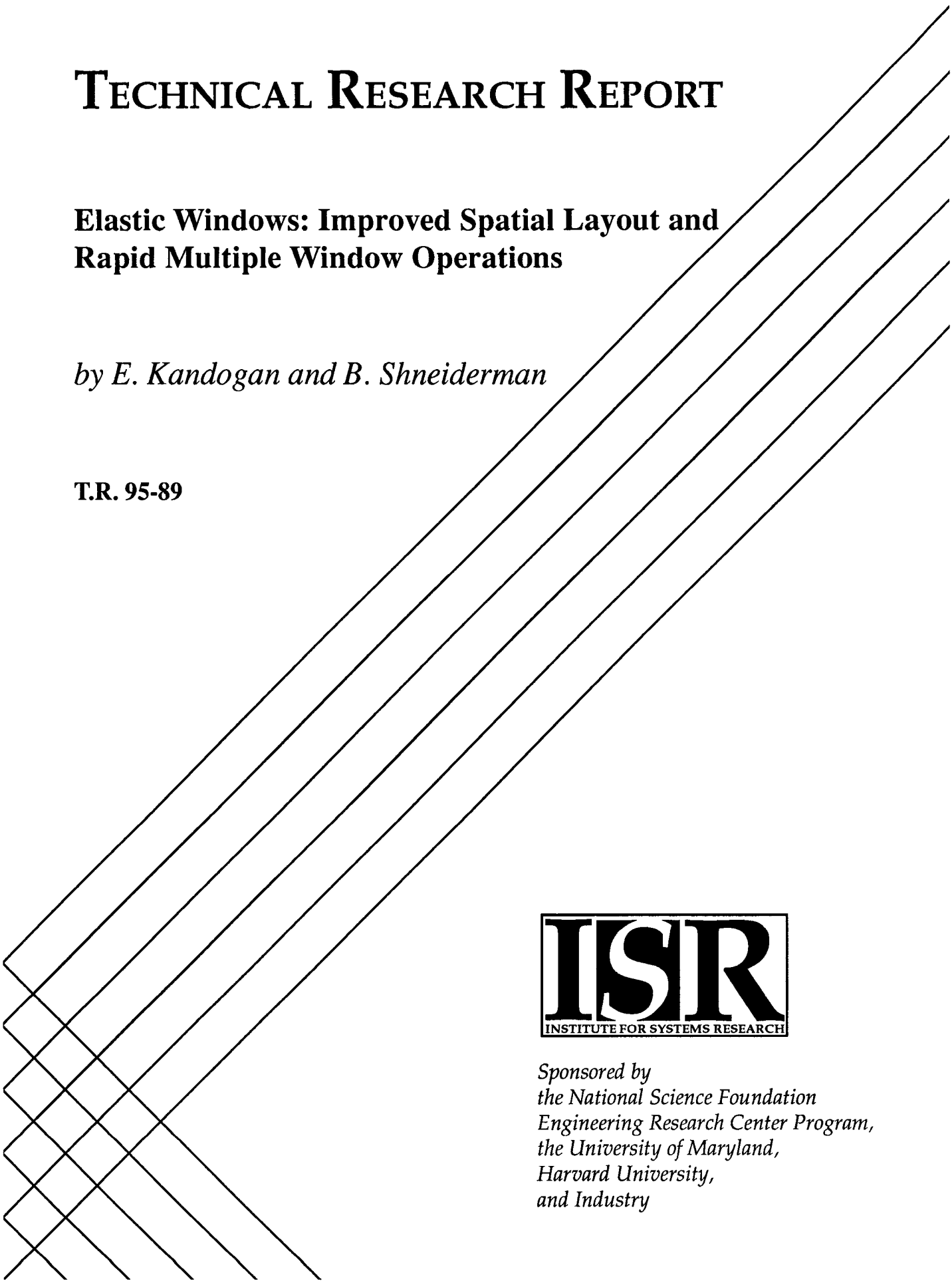
TECHNICAL RESEARCH REPORT

Elastic Windows: Improved Spatial Layout and Rapid Multiple Window Operations

by E. Kandogan and B. Shneiderman

T.R. 95-89



Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry

CAR-TR-786
CS-TR-3522
ISR-TR-95-89

Sept. 1995

Elastic Windows: Improved Spatial Layout and Rapid Multiple Window Operations

Eser Kandogan and Ben Shneiderman*

Human-Computer Interaction Laboratory
Center for Automation Research, Department of Computer Science
Institute for Systems Research*
University of Maryland, College Park, MD 20742-3255
kandogan@cs.umd.edu, ben@cs.umd.edu
Tel: (301) 405-2680

Abstract

Most windowing systems follow the independent overlapping windows approach, which emerged as an answer to the needs of the 80s' applications and technology. Advances in computers, display technology, and the applications demand more functionality from window management systems. Based on these changes and the problems of current windowing approaches, we have updated the requirements for multi-window systems to guide new methods of window management. We propose elastic windows with improved spatial layout and rapid multi-window operations. Multi-window operations are achieved by issuing operations on a hierarchically organized group of windows in a space-filling tiled layout. Sophisticated multi-window operations like Hook, Pump, Minimize, Restore, Move and Relocate have been developed to handle fast task-switching and to structure the work environment of users to their rapidly changing needs. We claim that these multi-window operations and the tiled layout decrease the cognitive load on users. Users found our prototype system to be comprehensible and enjoyable as they playfully explored the way multiple windows are reshaped.

Keywords: Window Manager, CAD, Task Switching, Multi-window operations, Personal Role Manager, Programming Environment, Navigation Problem

The Human-Computer Interaction Laboratory (HCIL) is an interdisciplinary effort within the Center for Automation Research. The main participants are faculty, staff, and students from the Department of Computer Science, Department of Psychology, and College of Library and Information Services at the University of Maryland, College Park, MD.

http://www.cs.umd.edu/projects/hcil/

ftp ftp.cs.umd.edu/pub/hcil

email hcil-info@cs.umd.edu

For single technical reports or information not available at the ftp site or our url please write to:

Janet Sumida
Human-Computer
Interaction Laboratory
A.V. Williams Building
University of Maryland
College Park MD 20742

INTRODUCTION

It is widely believed that windowed environments are superior to non-windowed ones. However, an early study by Bury et al. [4] (1985) comparing users' performance in windowed systems to non-windowed systems revealed that task-completion time in windowed systems can be longer due to window arrangement time. A detailed analysis, however, showed that actual times spent on solving a task were lower in windowed environments compared to non-windowed environments. Their experiments also showed that the error rates in windowed environments were significantly lower. Although systems compared in these experiments were rather old, the results clearly indicate that benefits of windowing can be overshadowed by the extra time spent on window housekeeping activities.

Card et al. [5] identified seven functional uses of multiple windows. Among these, independent control of multiple programs, referred to here as multitasking, is the most significant. Basically, it is the ability of users to work on different tasks in separate windows. Analyses of work flow determined that people deal with many tasks concurrently with frequent switches between tasks [2]. For example, a researcher preparing a paper might draw the figures in one window while writing the text of the document using an editor in another window. Multitasking results in improvements on the overall user performance due to the decreased average task-completion time. Windowing systems must provide good mechanisms for task-switching to make multitasking more beneficial.

Windowing is also useful in the case of a single task. It is possible to reduce the cognitive load on users by allowing them to examine other windows for supplementary information for the task at hand or use task-aids like cut-and-paste.

As stated by Card et al. [5], the computer display is used not only as a communication medium but also as an external memory for users. Thus having all the necessary information on the screen and filtering out unnecessary windows is a required property of windowing systems. Malone [12] observed that the way people organize papers on their desk helps them to structure their work and reminds them of unfinished tasks. As Funke et al. [9] suggested, windowing systems should support users to integrate, organize, compare, distill, summarize, and apply the information.

Today's windowing systems do not differ much in their basic principles of window management. Almost all systems follow the independent overlapping windows approach, where windows are allowed to overlap each other, operations on windows are performed one at a time, and size and location of each window is independent.

The independent overlapping windows principle emerged as an answer to the needs of 80's applications and technology. 80's applications were mostly single-window applications, with all the information related to a task in one window. Thus, it was preferable to have independent size and location for each window/task. Since these applications were single-window applications, task-switching was not a big problem. When users wish to continue with the task previously abandoned, the window for that task has to be found on the screen and brought to the front.

With the typical early 80's display resolution (640×480) it was not possible to display two page-sized documents on the screen simultaneously. Overlapping windows came as a solution to the small-screen problem by allowing more windows to be open simultaneously.

Resolutions like 1280×1024 are quite common these days, which is roughly four times the 80's resolution. Besides the resolution, display speed increased as well, which made sophisticated animations feasible. Animations in windowing systems help users to understand the result of operations and decrease the cognitive load.

With advances in computer technology, more demanding applications come into existence. Computer-Aided Design (CAD), and Computer-Aided Engineering (CAE) are typical multi-window applications. In these applications, it is typically necessary to open many windows displaying simultaneously different parts or representations of the system under design. With the increase in the number of windows, visualizing simultaneously all the necessary information for a task became difficult. As the number of windows per task increases, task-switching becomes more time-consuming since more windows need to be opened/closed or moved/resized under the independent overlapping windows approach. Due to the independence of windows, each window must be operated separately. Longer delays due to housekeeping further increase task-completion time because of the loss of users' mental task context, which implies a non-linear cost curve as the number of windows per task increases. Contents of short-term memory are not only affected by the time that passes, but also by the type of work carried out during that time period. Window housekeeping is an activity related to the computer domain and not to the users' task [17]. Thus, time spent on window management substantially increases the disruptive effect on the short-term memory.

Multiplicity of actions is one mechanism to improve performance. Use of regular expressions, wild-cards, and aliases are some of the ways to accomplish multiplicity in traditional command languages. In recognition of the need for multi-window operations, some windowing systems, like X Windows, introduced a limited parenthood relationship. With a single action families of windows can be opened or closed. However, the multiplicity of window operations in current systems is limited.

Bly and Rosenberg [3] characterized the requirements of multi-window systems as the ability of the windows to conform to their contents and the ability of the system to relieve the user of window management.

On the basis of the problems discussed, we have updated these requirements:

- support multi-window operations to promote organization and coordination of windows according to tasks.
- allow fast task-switching and resumption.
- free users' cognitive resources to work on task related operations rather than to window management operations.

- use screen space efficiently and productively for the tasks.
- allow fast temporary window arrangements.

Systems developed to address some of these requirements are described in the Related Work section at the end of the paper.

BASIC PRINCIPLES OF ELASTIC WINDOWS

Our method is based on three principles: *hierarchical window organization, space-filling tiled layout, and multi-window operations.*

Hierarchical window organization supports users structuring their work environment according to tasks. The hierarchical organization of windows allows users to map their task hierarchy onto the nested rectangle tree structure. Window operations can be applied at any level of the hierarchy, which makes multi-window operations possible. Operations are issued to the root of a subtree of windows, where changes due to the operation are propagated to lower-level windows in that subtree e.g. groups of windows can be minimized, and resized together. Typically, people organize papers on their desk as piles, and move all of them simultaneously. Hierarchical organization and applicability of window operations at any level allow rapid task-switching, even when the number of windows is large.

We have chosen the tiled window layout as our window organization style. Although it is possible to present windows visually as a hierarchy in the overlapping window layout using some form of cascading, we have found tiled layouts more suitable for the purposes of satisfying the requirements. In tiled layouts, hierarchies of windows can be easily represented by the borders surrounding the subwindows. Subwindows at the same level in the hierarchy can be placed either horizontally adjacent, vertically adjacent, or mixed (Figure 1). This feature allows some flexibility in the placement of windows under the same hierarchy and allows windows to conform to their content. The content of windows is an important constraint on which users determine the shape and size of windows.

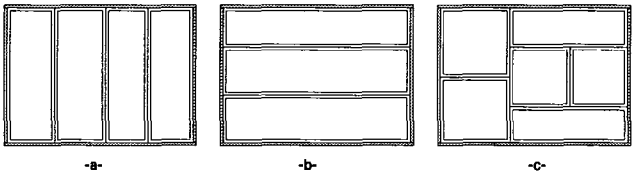


Figure 1: Example layouts of sub-windows in a two-level hierarchy: a) Horizontally adjacent b) Vertically adjacent c) Mixed

As Cohen et al. [7] stated, overlapping window layouts are difficult to handle when large numbers of windows must all be visible at once, and they come and go rapidly. Tiling is especially useful for applications that generate a large number of short-lived windows [7].

We have taken a space-filling tiled approach, called *elastic windows*, in order to use screen space productively, avoiding the wasted background of the overlapped windows approach. Groups of windows stretch like an elastic material as they are being resized, and other windows shrunk to make space.

We claim that multi-window operations decrease the load on the cognitive abilities of users by decreasing the number of window operations. Operations like multi-window open, close, hook, pump, minimize, restore, move, and relocate enable users to change the window organization quickly to compare, filter, and apply the information.

Resize operations like hook and pump help the user to compare information in separate windows; move and relocate operations enable fast organization of windows to quickly changing demands; minimize and restore operations help the user to filter-out unnecessary information as well as enabling fast task-switching. Our system takes advantage of spatial memory, which is important for remembering the window contents. Thus minimized windows are shown in the same position with reduced size. Minimized windows also remind users of unfinished tasks. It is possible to restore a group of windows rapidly and easily by a single operation to their previous sizes. The Restore operation helps users to reconstruct their previous working environments easily.

THE ELASTIC WINDOW

In addition to the window contents, each elastic window consists of borders surrounding the content area from four sides, a title on the upper border, and a gadget to the left of the title.

The gadget is used to invoke a menu for some of the window operations, whereas the borders are mainly used for resize operations. Basically, the border is dragged using the mouse, until the appropriate size is reached. Immediate visual feedback is provided during the operation using animations that slowly stretch the border. The corners of the border are used for diagonal resizing, while the rest of the border is used for one-dimensional resizing.

Borders are also used to indicate hierarchical groupings of windows. Border coloring gradually changes according to the level of the window in the hierarchy to make groupings recognizable. Border thickness is important since deep nesting may result in more space being used for borders instead of useful information. During our design, we have found that borders as thin as 3-4 pixels are easily operable.

The effect of changes in window size on the content depends on the application. For example, upon down-sizing a window used for viewing a document, it might be preferable to see the same content but with smaller font sizes; but when designing a system in a CAD system, keeping the same zooming factor and clipping might be preferable. When clipping is used, facilities like scrollbars are needed to move the viewing area. Similar arguments can be made for other content types like images and icons. The choice is made by the application program, based on users' preference. The action is initiated when the window manager sends a window size update.

Only windows at the leaf level contain information. Windows at higher levels are containers for their children windows.

Users can set the minimum window size. Even when the window is so small that its contents are not fully visible, it still gives users some information about its content because of the spatial placement and reminds users of unfinished tasks; and it can be enlarged rapidly and easily if needed.

ELASTIC WINDOW OPERATIONS

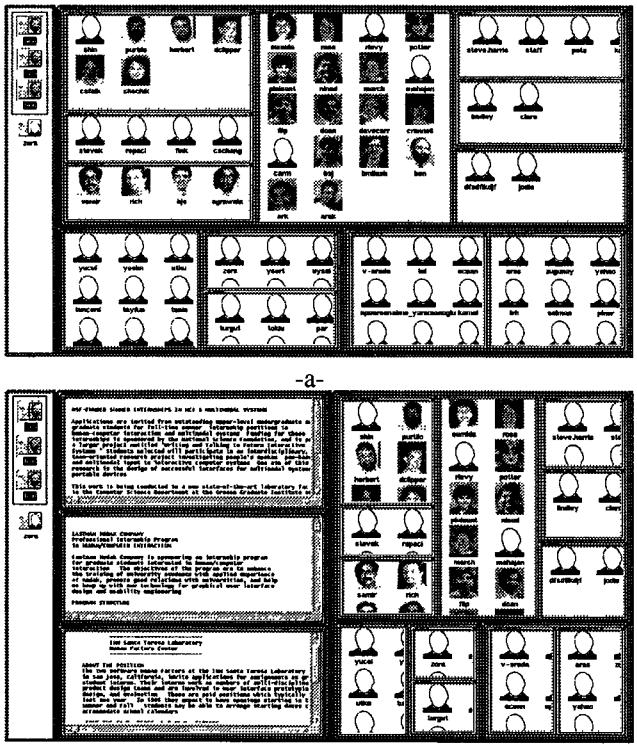
The elastic window operations, which allow simultaneous changes to multiple windows are:

- Open/Close
- Resize
- Minimize/Restore
- Move/Relocate

Open/Close Operations:

With the open operation, the window contents can be determined either prior to or after the operation. In the former case, the window contents, selected before, are displayed as soon as the window is opened, whereas in the latter case an empty window is opened, which can be used either as a container for subwindows or its contents can be filled by users.

A new window can be opened by double-clicking on the border of an existing window or by selecting from the menu. Double-clicking on a border causes the existing window to be pushed according to the position of the border to open space for the newly created window.



-a-

Figure 2: Mail-tool example: a) Initial screen b) Multi-open operation

Select-drag-drop can be used to achieve multi-open operation. First, data objects are selected from other windows, dragged, and then dropped in an empty window, causing as many windows, all in one group, to be opened as the cardinality of the selection. Each such window contains some information related to the data object e.g. a detailed view or another representation. Since all these windows are opened in one group, it is also possible to close all of them at once. For example, this feature can be used in the mail-tool application (Figure 2) to view all incoming mail messages from a person with a single operation. The new messages are shown iconized in the left window. Old messages are displayed as icons, grouped hierarchically in separate windows on the right. First, the user opens an empty window to the left of the OldMail Window by double-clicking on the left border of the OldMail Window. The available space is partitioned to windows at the same level according to their sizes before the operation. The new window has the same size as the OldMail window. When the user selects the icons, representing incoming messages, from the NewMail Window, drags and drops them into the empty window all of the windows containing messages are opened at the same time. Comparison of the elastic windows approach with the independent overlapping windows approach in terms of the number of operations in this example is shown in Table 1.

Approach	Resize	Open	Move
Independent	17	3	>9
Elastic	1	1	0

Table 1: Comparison of the number of window operations in the Mail-tool example

When the Open operation is selected from the menu, a subwindow is opened inside the existing empty window. This way hierarchical windows can be created on the fly.

A window is closed by selecting the Close operation from the menu. When a window is closed, the freed space is partitioned to other windows at the same level proportional to their previous sizes. The Close operation can also be applied to windows at any level of the hierarchy. Closing a higher level window will close all its subwindows as well.

Resize Operations:

There are two kinds of resize operations: Pump and Hook. While the effect of the Pump is time-dependent, the Hook operation requires the user to drag on the border of a window.

The Pump operation can be invoked by selecting Pump from the menu and then pressing either the left or right button of the mouse either on the border or inside the content area of a window. Pumping of windows at higher levels in the hierarchy can be done by pressing only on the border. Pressing the left (right) button causes window size to be enlarged (reduced) in all directions according to the duration of press.

Hook operations are activated by pressing a mouse button on the border and then dragging the mouse until the appropriate size is reached. Both unidirectional and bidirectional hook are possible which work either horizontally, vertically, or diagonally. In bidirectional hook, both the borders resize by the same amount.

Resize operations also affect other windows on the same level of hierarchy. They result in either a push or pull depending on the border dragged and the direction of drag (Figure 3). In 3.a, Window C pulls windows A and B, since the left border of Window C is dragged to the right. In 3.b, Window B pushes windows C, D, and E, since the right border of Window B is dragged to the right. Windows not affected are grayed in the figure.

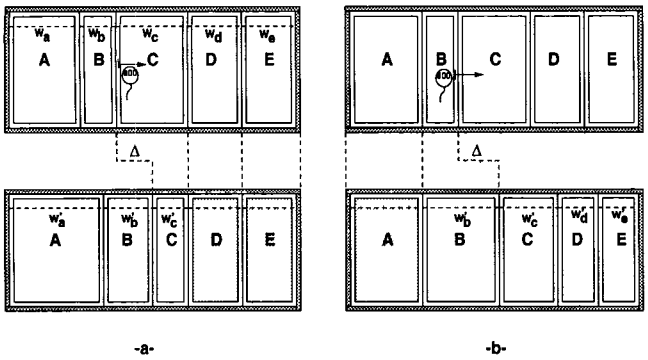


Figure 3: Effect of resize operations on other windows: a) Pull effect b) Push effect

When the size of a higher level window changes, the effects of that change are propagated down the root of the subtree to lower-level windows. New sizes for these windows are calculated proportional to their previous sizes. Users of our prototype system found these operations to be comprehensible and even fun.

Minimize/Restore Operations:

Windows at any level of the hierarchy can be minimized by selecting from the menu. Windows minimized appear in the same location, but with only their title shown in a rectangular region. Figure 4 shows the results of three Minimize operations made on the original layout in 4.a. Figure 4.b is an example of horizontal minimization, whereas 4.c is an example of vertical minimization. Figure 4.d shows the result of the minimize operation applied to windows at a higher level.

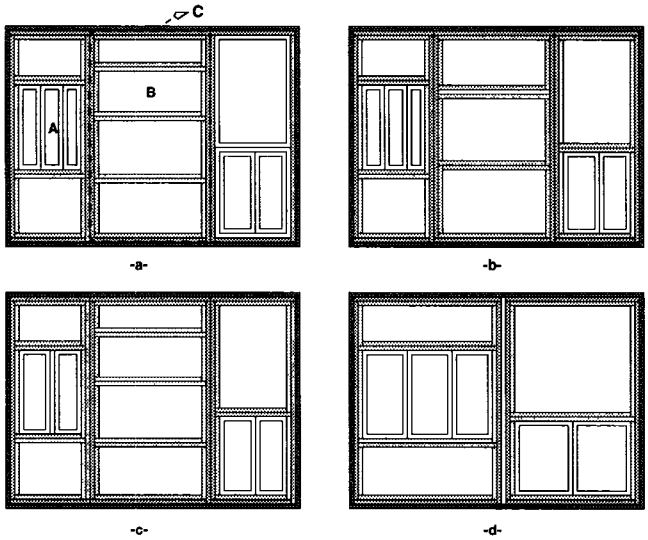


Figure 4: Minimize operations: a) Initial Layout b) Window B minimized c) Window A minimized d) Window C minimized

A minimized window can be reopened with its previous size by the Restore operation. The Restore operation is invoked by double-clicking on the minimized window.

The Minimize operation is primarily used to abandon a task for a while and open up space for other tasks. The minimized window can be reopened with a single Restore operation with the size before the operation. Hence, Minimize/Restore operations allow fast task-switching and resumption. A similar effect could be achieved by resizing the windows to very small sizes. Later, these windows can be resized to their original size to continue with the work abandoned.

Move/Relocate Operations:

The Move operation changes the position of a window or hierarchy of windows without changing the size. This operation can be visualized as shifting a window without changing its position relative to its siblings. The Move operation is accomplished by dragging with the middle button pressed (Figure 5.a).

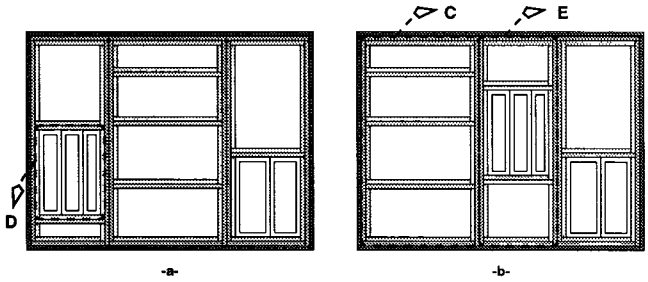


Figure 5: Move/Relocate operations: a) Window D moved down b) Window E relocated to the right of Window C

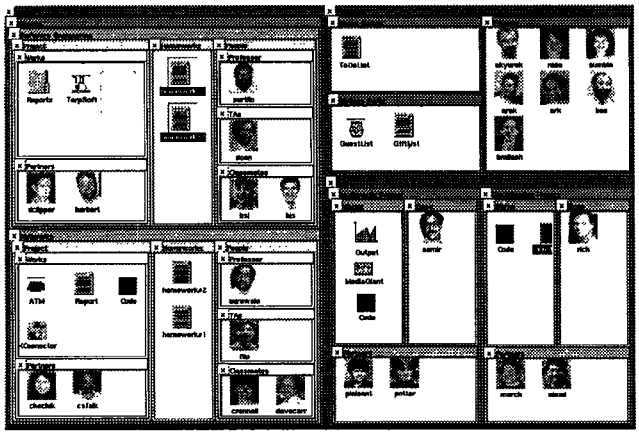
The Relocate operation is used to relocate a window or group of windows to any position in the hierarchy. The Relocate operation is accomplished by first selecting from the menu and then double-clicking on a border as in the Open operation (Figure 5.b). Minimized windows can be moved and relocated as well.

SCENARIOS

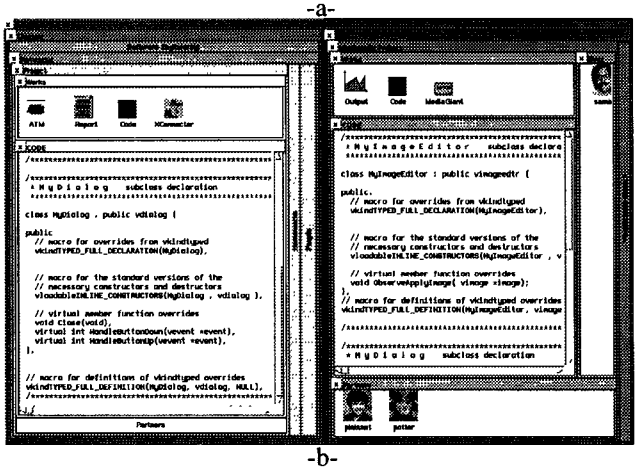
Personal Role Manager:

The Personal Role Manager (PRM) provides users with a role-centered environment, where people can structure the screen layout and the interface tools to match their roles [18]. The goal is to simplify and speed the coordination of tasks. Thus, fast access to partners, schedules, tools, and documents regarding each role, and fast switching between roles is a requirement of PRM.

Elastic windows can be used in the design of PRMs. In Figure 6, two snapshots of the PRM of a student are shown. This student has a number of other roles like the organization of a birthday party, home duties and job responsibilities. As shown in Figure 6.a, this student is working on two projects at work, and takes two classes this semester. Documents, partners, and tools for each role are displayed in a hierarchical organization. Windows for the student role are on the left, while windows related to the job are on the right below the windows for birthday organization and home duties. Comparison of the elastic windows approach with the independent overlapping windows approach in terms of the number of operations is shown Table 2.



-a-



-b-

Figure 6: PRM Example: a) Initial Screen b) Focus on class report and project

Approach	Resize	Open	Minimize	Move
Independent	6	2	16	>6
Elastic	2	2	4	0

Table 2: Comparison of the number of window operations in the PRM example

Window layout can give an overview of roles, as well as be customized according to personal preference. In Figure 6.b, the layout has been customized to enable the student to reference the code in Network class, while working on the code for Multimedia project at work.

With the use of multiple window operations, as well as resize, minimize, and restore operations, users can focus on their roles rather than arranging windows. Fast switching among roles enables users to work at their own pace, with minimum distraction due to window housekeeping.

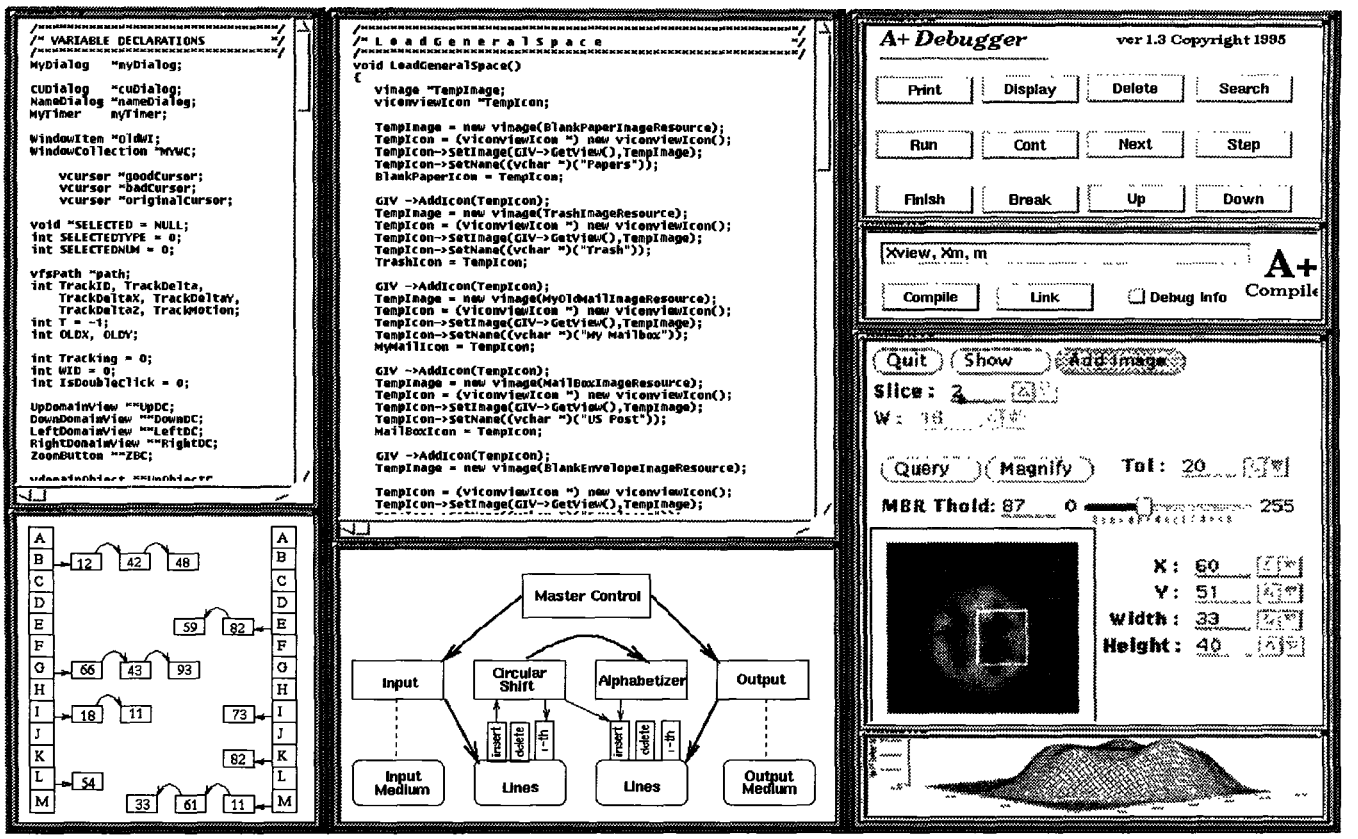
Programing Environment:

Programmers typically need to reference different parts of the code, such as data declarations, procedure declarations, bodies, invocations, and the main program as well as the documentation regarding its structure like charts, and execution diagrams. The Edit-Compile-Run-Debug cycle is repeated many times during program development. During the Edit phase, when some part of a program needs to be referenced, typically programmers search to find the right place. When considering a change that would affect many parts of the program, cognitive effort is exerted to remember how that change might affect different parts of the code. Since managing many windows is difficult, typically two windows are used, for example one containing the procedure body, and another, called the reference window, to view the invocations of that procedure in different parts of the program. The search in the reference window is made sequentially forcing the programmer to remember code seen earlier while deciding on the effects on the current position.

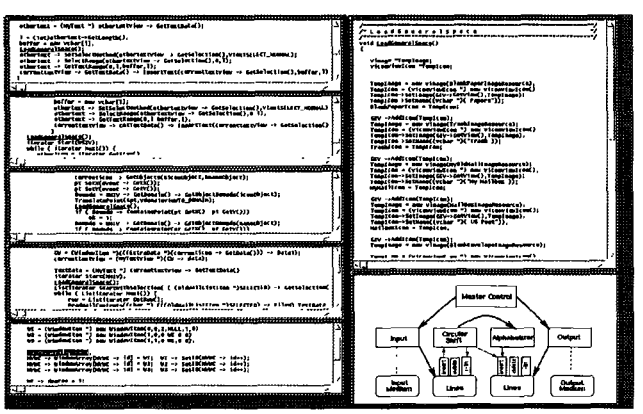
With the use of multi-window operations, it may be possible to improve the efficiency of program development. At this phase of the development of our system, only the window operations have been implemented with no connections to applications. To better describe the uses of the system, we inserted the images of the debugger, compiler, and charts into windows in Figure 7.

Initially, the screen contains windows for the code of the data declarations with its chart below, the code of a procedure above the organization diagram of the program, and debugger and compiler windows above the window of the application with its output (Figure 7.a). When an error is detected in a procedure during debug phase, a multi-open operation can be issued to view all the invocations of that procedure. There are a number of ways to do this. One of the possibilities is to click on the procedure name which pops up a menu of functions. Upon selecting the *Open Invocation Locations* function, all the segments of the program calling that procedure are placed in separate windows all in one group (Figure 7.b). This layout helps users in making the decisions concerning the change, since all five references are shown on the screen at the same time. The windows containing the code for the data declarations and its chart have been minimized. Since both of these windows are in one group, only a single Minimize operation is made. Similarly debugger, compiler, application and output windows are all minimized with a single Minimize operation for better visibility. It is possible to concentrate on each of these invocation locations one at a time, by the use of bidirectional vertical hook (Figure 7.c). Switching interest between these windows is possible by the use of the resize operations. Once the decisions concerning the change have been made, all reference windows can be minimized with a single operation since they are all in the same group. Later, when editing the planned changes in the procedure body part, all these reference windows can be reexamined by a single Restore operation.

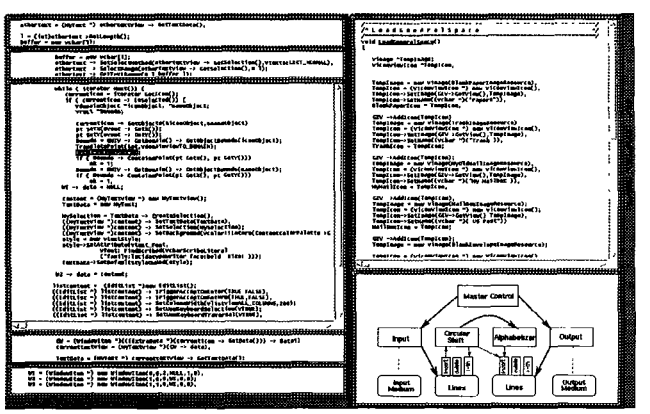
By the close interaction of an application with the window manager as in the multi-open in this example, dramatic improvements can be achieved in the users' performance. This example also demonstrates the use of multi-window operations to reduce the burden of window management and the use of the Minimize/Restore operations to allow easy task-switching.



-a-

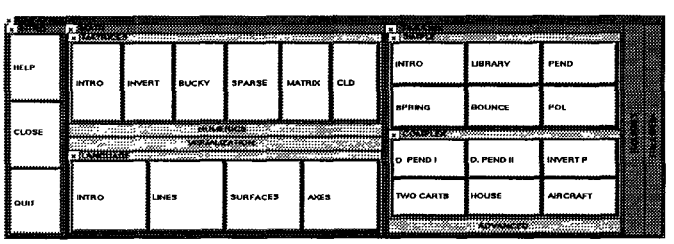


-b-

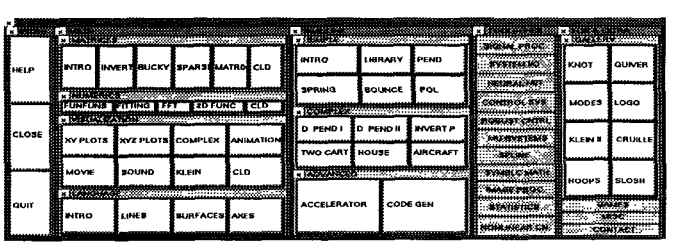


-c-

Figure 7: Programming Environment Example: a) Initial Screen b) Multi-open operation to see invocations c) Bidirectional vertical Hook to create larger window



-a-



-b-

Figure 8: An Example Hierarchical Menu Organization: a) Layout for novice use b) Layout displaying advanced menu items for frequent users

Menu Organizer for Complex Applications:
A typical problem in complex applications with hundreds of menu items is the organization of these items to enable fast selection. In complex applications, users also have difficulties in understanding the set of available and/or currently applicable items. The usual nested menu structure falls short of giving the user an overview of items. As the number of items increases, it becomes hard to find the path to an item. This is commonly referred as the navigation problem. Short-cut key combinations come as a solution, but they are useful only for expert users. The problem in the nested menu structure is caused by the fact that only the highest level items and the current path are displayed.

Our design principles are also applicable to this problem. With the hierarchical layout, it is possible to give the user an overview of the menu items available in the system. Using color coding, the current set of applicable items can be shown. With the use of Minimize and Restore operations, users can configure the organization of the items according to their preference and the task. Thus, commonly used items can be accessed with a single click no matter how deep in the hierarchy they are. Figure 8 shows two snapshots of the hierarchical organization of 97 items in a commercial CAE package.

RELATED WORK

The Rooms system [10] uses multiple virtual workspaces, where the overlapping window strategy is used in each of these single-screen workspaces. Each task is devoted to a workspace, where users can switch to other tasks using either the overview or the doors between workspaces for rapid transitions. Basically, the Rooms system tries to overcome the problems due to the increase in the number of windows by increasing the total screen space, by introducing multiple virtual workspaces, and by techniques which allow fast switching between workspaces. Also, it allows users to organize tasks into workspaces, where all windows belonging to a single task exist. Windows belonging to a task are restricted to fit in a single screen. Although it is possible to partition tasks into subtasks and place each subtask in different workspaces and utilize doors for efficient transitions between these workspaces, users can easily lose task context since information for a task is distributed to multiple screens. There is no mechanism which allows multi-window operations. Tasks are restricted to fit in a two-level hierarchy: the overview level, and the workspace level.

RTL/CRTL [7, 8] uses constraint-based tiled window strategy without hierarchy. Window management is based on constraints on the presence, size, location, adjacency, alignment of windows, and degree of automation. Constraints generally make systems harder to learn; simple general rules with no exceptions are preferable.

CIWM [9] uses automated window management for window creation, sizing, placement, removal, and organization, though user overrides are possible. The hybrid window layout strategy is used, where the system tries to position windows with no overlaps. In certain situations, however, overlaps are allowed. Although automatic strategies in window management relieve the burden of window management, direct user control is preferable as in most HCI artifacts.

Xerox/Star and Windows 1.0 also used tiling, but hierarchical organization and multiple operations were not provided.

CONCLUSION

We have attempted to determine the extended requirements of multi-window systems adjusted for today's applications and technology. Characteristics of modern applications demand more functionality than what is available in today's windowing environments. Multi-window operations, organization of windows by tasks, and capability to handle frequent task-switching without demanding extensive cognitive abilities are some of the requirements of future windowing systems.

Elastic windows is a space-filling hierarchical tiled approach that we believe satisfies the requirements. A prototype has been developed with the window operations and the organization of windows by tasks implemented. More work is needed on the interaction with the applications to make it a general window manager. Coordination of windows by task, like synchronized scrolling, hierarchical browsing, and direct selection, will be studied. Both the usability of the window management operations and their comprehensibility need experimentation, but users are attracted to the playful animations of elastic window interactions. The use of elastic window management method as a navigation technique for hierarchical menus warrants further research.

ACKNOWLEDGEMENT

We appreciate comments from Catherine Plaisant during the project. We are grateful to Kent L. Norman, Charles Goodrich, Gary Marchionini, Khoa Doan, Brett Milash, Kasim S. Candan, and Egemen Tanin for their comments on the draft of this paper. This research is supported by a grant from the National Science Foundation under Grant No. NSF EEC 94-02384.

REFERENCES

1. Asahi, T., Turo, D., Shneiderman, B., Using treemaps to visualize the analytic hierarchy process, *to appear in Information Systems Research*, (Sept 1995).

2. Bannon, L., Cypher, A., Greenspan, S., Monty, M. L., Evaluation and Analysis of Users' Activity Organization, *Proc. of the CHI'83, Human Factors in Computing Systems Conference*, ACM, New York, NY, (1985), pp. 54-57.

3. Bly, S., Rosenberg, J., A comparison of tiled and overlapping windows, *Proc. CHI '86 Conference - Human Factors in Computing Systems*, ACM, New York, NY, (1986), pp. 101-106.

4. Bury, K. F., Davies, S. E., and Darnell, M. J., Window management: A review of issues and some results from user testing, *IBM Human Factors Center Report HFC-53*, San Jose, CA, (June 1985), 36 pages.

5. Card, S. K., Pavel, M., and Farrell, J. E., Window-based computer dialogues, *INTERACT '84, First IFIP Conference on Human-Computer Interaction*, London, UK, (1984), pp. 355-359.

6. Card, S. K.,Henderson, A., A multiple virtual-workspace interface to support task switching, *Proc. CHI '87 Conference - Human Factors in Computing Systems*, ACM, New York, NY, (1987), pp. 53-59.

7. Cohen, E. S., Smith, E. T., Iverson, L. A., Constraint-based tiled windows, *IEEE Computer Graphics and Applications 6*, 5, (May 1986).

8. Cohen, E. S., Berman, A. M., Biggers, M. R., Camaratta, J. C., Kelly, K. M., Automatic strategies in the Siemens RTL tiled window manager, *Proc. IEEE 2nd International Conference on Computer Workstations*, IEEE, Piscataway, NJ, (1988), pp. 111-119.

9. Funke, D. J., Neal, J. G., Paul, R. D., An approach to intelligent automated window management, *International Journal of Man-Machine Studies 38*, (1993), pp. 949-983

10. Henderson, A., Card, S. K., Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface, *ACM Transactions on Graphics 5*, 3, (1986), pp. 211-243.

11. Lifshitz, J., Shneiderman, B., Multi-window browsing strategies for hypertext traversal, *Proc. 30th Annual Technical Symposium of the Washington, DC Chapter of the ACM*, (1991), pp. 121-131.

12. Malone, T. W., How do people organize their desks? Implications for the design of office automation systems, *ACM Transactions on Office Information Systems*, 1, pp. 99-112.

13. Myers, B., Window interfaces: A taxonomy of window manager user interfaces, *IEEE Computer Graphics and Applications 8*, 5, (September 1988), pp. 65-84.

14. Norman, K. L., Weldon, L. J., Shneiderman, B., Cognitive layouts of windows and multiple screens for user interfaces, *International Journal of Man-Machine Studies 25*, (1986), pp. 229-248.

15. Plaisant, C., Shneiderman, B., Organization Overviews and Role Management: Inspiration for Future Desktop Environments, *Proc. IEEE 4th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, (April 1995).

16. Plaisant, C., Carr, D., Shneiderman, B., Image browsers taxonomy and design guidelines, *IEEE Software 12*, 2, (March 1995), pp. 21-32.

17. Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction: Second Edition*, Addison Wesley Publ. Co., Reading, MA, (1992), Ch.9.

18. Shneiderman, B., Plaisant, C., The Future of Graphic User Interfaces: Personal Role Managers, *People and Computers IX*, Cambridge University Press, (1994)