Elastic Windows: Improved Spatial Layout and Rapid Multiple Window Operations

Eser Kandogan

Department of Computer Science &
Human-Computer Interaction Laboratory
University of Maryland
College Park, MD 20742
Tel: (301) 405-2725
kandogan@cs.umd.edu

Ben Shneiderman

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

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 window groups hierarchically organized in a space-filling tiled layout. Sophisticated multi-window operations and spatial layout dynamics helps users to handle fast task-switching and to structure their work environment to their rapidly changing needs. We claim that these multi-window operations and the improved spatial 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, Elastic Windows

INTRODUCTION

It is widely believed that windowed environments are superior to non-windowed ones. However, an early study by Bury et al. [5] (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.

to appear in ACM AVI'96 Advanced Visual Interfaces, Gubbio, Italy, May 27-29.

Card et al. [6] analyzed window usage according to tasks and 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 among them [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 allows access to multiple sources of information. It is possible to reduce the cognitive load on users by allowing them to examine other windows for supplementary information, or multiple representations for the task at hand or use task-aids like cut-and-paste.

As stated by Card et al. [6], 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 [16] 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. [11] suggested, windowing systems should support users to integrate, organize, compare, distill, summarize, and apply the information.

MOTIVATION

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.

With the typical early 80's display resolution (640 $\times$ 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 $\times$ 1024 are quite common these days, which is roughly four times the 80's resolution. Besides the resolution, graphics processing 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. The amount and variety of information that users have to deal with increased a lot with advances in networks and the Internet. The information that is facing the users is usually unorganized and dynamically changing, thus users themselves need to do the organization. Typically when exploring information users want to keep both detail and overview.

Computer-Aided Design (CAD), Computer-Aided Engineering, Object-Oriented Development Environments, and Geographic Information Systems (GIS) are typical multi-window applications. In these applications, it is typically necessary to open many windows displaying simultaneously different parts or representations. Also opening separate windows for toolboxes, commands, and options is becoming the practice in complex applications.

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 handled separately. Longer delays due to housekeeping further increase task-completion time because of the loss of users' mental task context, kept in short-term memory. Increase in the number of windows also prevents users to see the overview of their desktop due to overlapping windows. This might delay users to switch to unfinished tasks.

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. Since window housekeeping is an activity related to the computer domain and not to the users' task [21], the time spent on window management substantially increases the disruptive effect on the short-term memory, thus implies a non-linear cost curve as the number of windows per task increases.

Gaylin [12] observed that the number of window operations that are used to switch the active window set constitutes 63% of all the operations in an independent overlapped window manager. This result supports the findings by Bannon et al. [2] that people switch among tasks frequently forcing them to change the visible set of windows on the screen.

Bederson and Hollan [3] observed that in traditional window-based systems there is no graphical depiction of the relationship among windows even when there is a strong semantic relationship. This problem is most apparent in hyper-text browsers and CAD systems, where each subwindow is either a link followed or part of the system under design. In current approaches, users have to deal with each window separately when organizing their desktop.

Kahn et al. [14] observed a similar phenomenon and called the presence of too many open windows "Windowitis". They observed that in Windowitis situations the users become quickly disoriented, lose the relationships that exist between windows due to loss of spatial cues, and become unproductive in completing their tasks.

Bly and Rosenberg [4] 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 users 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.
- provide a spatial layout that indicates the relationship between windows.

Earlier research that addresses some of these is described in the Related Work section at the end of the paper.

PROPOSED SOLUTION: ELASTIC WINDOWS

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

Hierarchical Window Organization:

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.

Figure 1 displays the mail-tool application written using elastic windows principles. The new messages are shown iconized in the left window. Old messages are displayed as icons, grouped hierarchically in separate windows on the right. The hierarchical layout clearly indicates the semantic relationship between the contents of the windows by the spatial cues in the organization of windows. The layout provides the user with an overview of all correspondence, where users can pick any category and work on it.

Multi-window operations:

Typically, people organize papers on their desk as piles, and move all of them simultaneously. Malone [16] found out that users like to group items spatially. We claim that providing multi-window operations on groups of windows can decrease the cognitive load on users by decreasing the number of window operations.

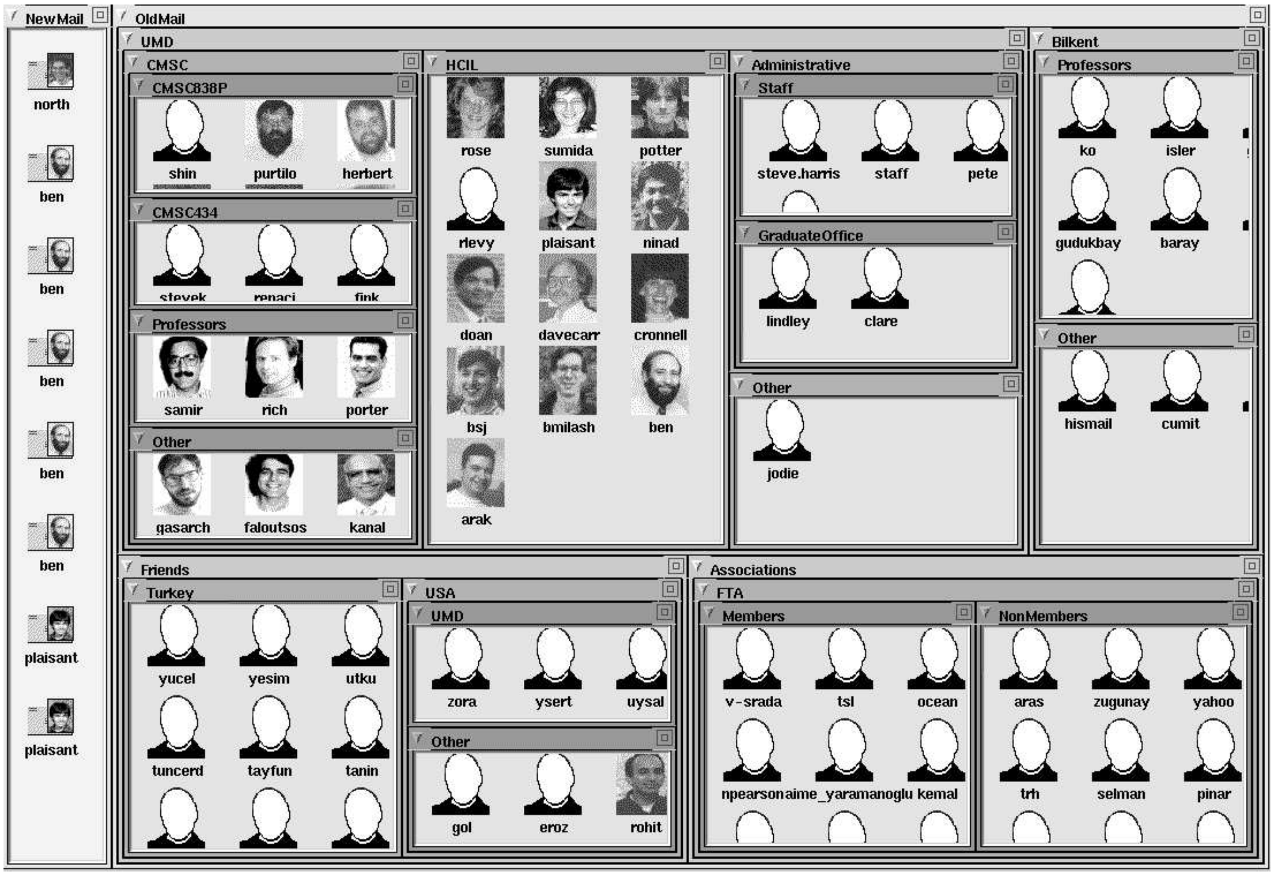

Figure 1: Mail-tool application: Organization of correspondence in a hierarchical layout gives the user an overview.

In elastic windows multiple operations are achieved by applying the operation to a group of windows at any level of the hierarchy. The results of the operations are propagated to windows inside that group recursively. This way groups of windows can be packed, resized, or closed with a single operation.

Another way to achieve multi-window operations is to select an operation and apply it to windows rapidly in a serial manner.

Operations like multi-window open, close, resize, pack, and unpack enable users to change the window organization quickly to compare, filter, and apply the information. Pack and unpack operations on groups of windows help users to filter-out unnecessary information as well as enable fast task-switching. Packed windows still appear in the same location preserving the spatial cues. This helps users to recall the window contents and reminds them of unfinished tasks. When a packed window is unpacked all the windows in the group are restored to their previous sizes, so that users can reconstruct their previous working environments easily.

Hierarchical organization and applicability of window operations at any level allow rapid task-switching, even when the number of windows is large.

Space-filling Tiled Layout:

We have taken a space-filling tiled approach 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 shrink to make space. Figure 2 shows an example resizing of the HCIL window under the UMD group window in the former example pushing the surrounding windows to the sides proportional to their sizes.

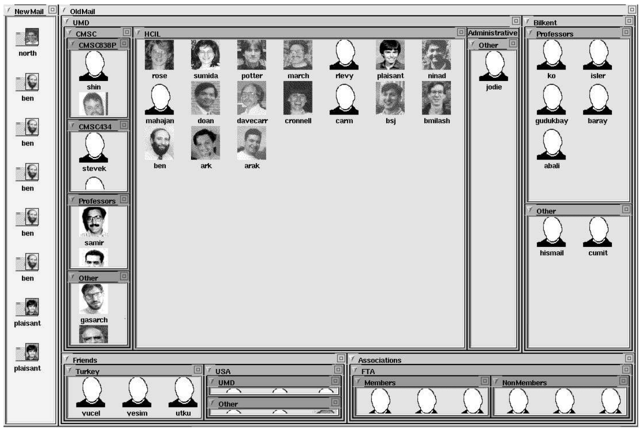

Figure 2: Elastic resizing of the HCIL window in the space-filling tiled layout of elastic windows.

We have chosen the tiled window layout as our window organization style in order to maximize the visibility of windows for a task. People typically try to organize windows to be non-overlapping while working on a task, even when overlapping windows are allowed. Other windows are left beneath the working set of windows.

As Cohen et al. [8] 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.

In tiled layouts, hierarchies of windows can be easily represented by the borders surrounding the subwindows. Users are quite flexible in the placement of subwindows in a group window. There is no strict horizontal or vertical placement rule. 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.

THE ELASTIC WINDOW DECORATION

In elastic windows window contents area is surrounded by borders on four sides. The top border is thicker than others, containing the title, a gadget to the left of the title, and a pump gadget at the rightmost position.

The left 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 as shown in Figure 1 according to the level of the window in the hierarchy to make groupings recognizable. Border thickness may result in more space being used for borders instead of useful information. During our design, we have found that borders as thin as 4-5 pixels are easily operable.

Pressing the left (right) button on the pump gadget causes the window size to be enlarged (reduced) in all directions according to the direction of the press.

Only windows at the leaf level contain information. Windows at higher levels are group windows containing subwindows.

LAYOUT DYNAMICS

Due to the space-filling tiled nature of the layout, window size changes affect size of other windows as well. In elastic windows the proximity of effect is limited only to windows under the same group and their subwindows.

Effect of the changes in the window size under the same group is split proportionally according to the window sizes. Depending on the border dragged and the direction of drag, it results in either a push or pull as shown in Figure 3.a and b. In both of these cases, window sizes are updated proportional to the sizes, but the set of windows affected changes.

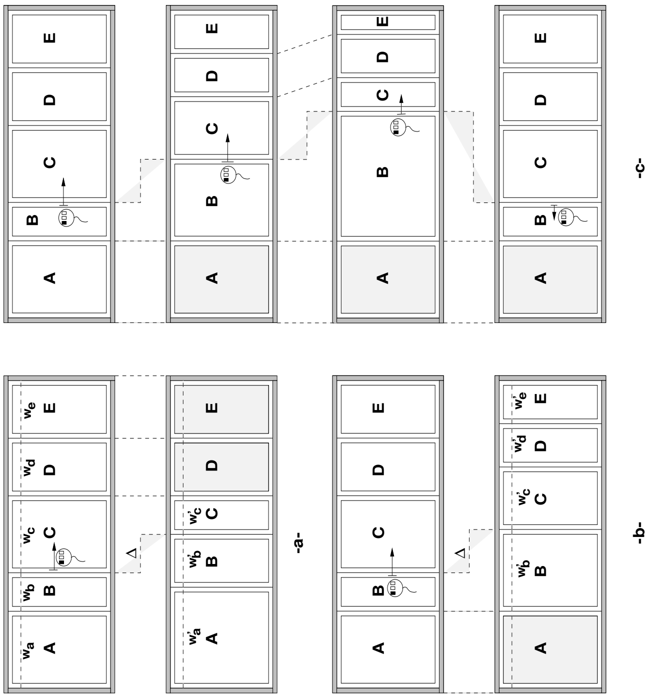

Figure 3: Effect of resize operations on other windows: a) Pull effect b) Push effect c) Recovering proportions on resize with minimum size windows.

Referring to Figure 3.a, Window C pulls windows A and B, since the left border of Window C is dragged to the right. In Figure 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. New window sizes are calculated as follows:

Pull	**Push**
$r = 1/(w_a + w_b)$	$r = 1/(w_c + w_d + w_d)$
$w'_a = w_a * (1 + \Delta * r)$	$w'_a = w_a$
$w'_b = w_b * (1 + \Delta * r)$	$w'_b = w_b + \Delta$
$w'_c = w_c - \Delta$	$w'_c = w_c * (1 - \Delta * r)$
$w'_d = w_d$	$w'_d = w_d * (1 - \Delta * r)$
$w'_e = w_e$	$w'_e = w_e * (1 - \Delta * r)$

Changes in the upper levels are propagated down to their subwindows recursively. For example, subwindows of Window A, B, and C in 3.a adjust their width accordingly.

Each elastic window has a default minimum window size, but users can set a different value for each window. This way users can protect a window from unwanted size updates.

When windows are being resized some of the windows may reach their minimum window size. For example, in Figure 3.b, when pushing windows C, D, and E, window D may reach its minimum size, while others don't. In that case the resize operation is allowed until all of the affected windows are fully compacted i.e. all reach their minimum sizes. Since window D is kept at its minimum size, proportions do change. However, in elastic windows the old proportions are kept, so that the resize operation is reversible. Figure 3.c shows an example.

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.

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 initiated by the window manager upon a window size update.

ELASTIC WINDOW OPERATIONS

In elastic windows, all window operations can be applied to individual windows as well as group of windows. Window operations supported are:

- Group
- Multiple Resize
- Multiple Slide/Relocate
- Multiple Open/Close
- Multiple Pack/Unpack
- Multiple Maximize

Creating Window Groups and Hierarchies:

Window groups can be created by opening an empty window by double-clicking on the border of an existing window. Dragging and dropping selected items in this empty window will open as many windows as the cardinality of the selection all grouped inside the empty window. Figure 4 shows the result of creating a window group for visualizing all new incoming mails from a person. Users can add more windows into a group later as well. A window can be removed from a group by simply closing that window.

Hierarchies of windows can be created similarly by opening an empty window inside another empty window. This is done by selecting the operation from the menu on the left gadget.

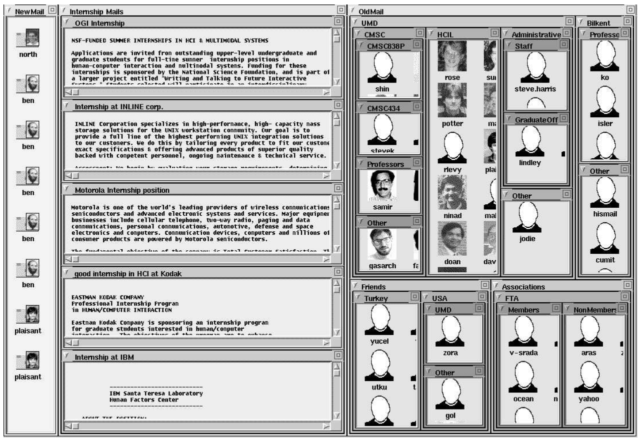

Figure 4: Creation of a window grouping visualizing new incoming mails from a person.

Multiple Open/Close Operations:

In elastic windows items selected are associated with windows. These windows generally contain some information related to the selected item e.g. a detailed view or another representation. The current system only supports iconic items and textual list items.

Once an item or group of items have been selected all of them can be opened with a single open operation by dragging and dropping on the border of an existing window. The existing window is pushed according to the position of the border to open space for the newly created window or windows. Double-clicking on the border achieves the same effect.

Selecting and opening a group of objects is primarily done to add a number of windows to an existing window group. This way multiple windows can be opened with a single operation.

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.

Multiple Resize Operations:

Windows at any level of the hierarchy can be resized by dragging on the border. All four borders of a window can be used in resizing. The drag direction and the border being dragged determines the effect as explained in the Layout Dynamics section.

The corners of a window is used for diagonal resizing. The sides are used for either horizontal or vertical resizing depending on the border.

Inside a group window with many windows open, typically users need to focus on one of these windows for a certain time. Bidirectional resizing becomes handy in such situations. This operation resizes a window in both directions by pushing/pulling the opposing borders by the same amount. Figure 5 shows the result of bidirectional resizing to see more of the contents of the middle window inside the group window.

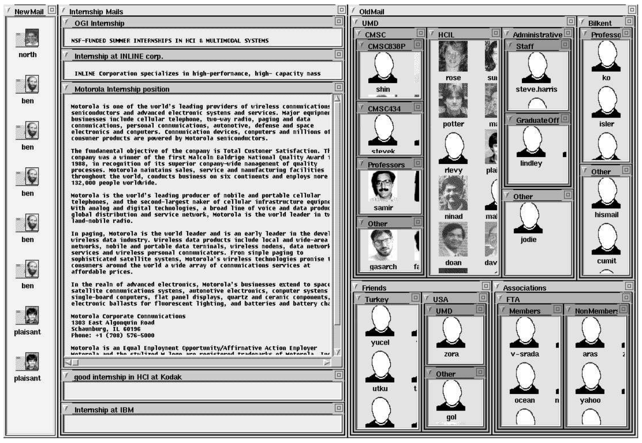

Figure 5: Bidirectional resizing is used to enlarge a window belonging to a group easily.

A window can be resized in all directions simultaneously by the pump operation. Pumping a window resizes the window pushing all the surrounding windows to the sides. The operation can be invoked by pressing either the left or right button of the mouse on the pump gadget. Pressing the left (right) button causes window size to be enlarged (reduced) in all directions according to the duration of press.

Multiple Pack/Unpack Operations:

Windows at any level of the hierarchy can be packed by selecting from the menu. Windows packed appear in the same location, but with only their title shown as a bar appropriately placed in the layout. This avoids the spatial disorientation which is typically the case in the iconify operation in most of the windowing system. In Figure 6, the group window used to view incoming mail messages is packed vertically, whereas the Administrative Window under UMD window is packed horizontally. The placement of the packed windows in the same position of the layout keeps the same spatial cues formed by the user when these windows are all open. This helps users to locate these windows easily.

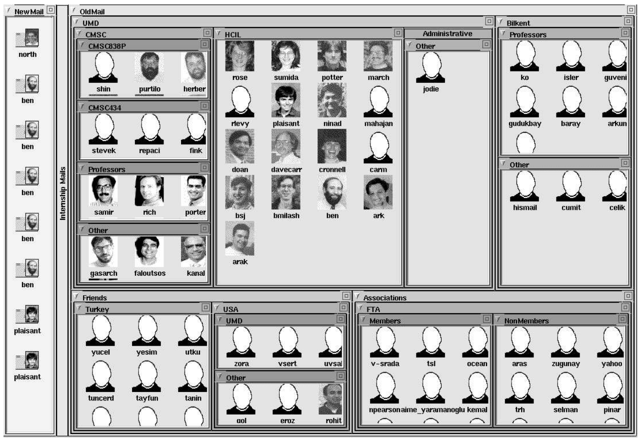

Figure 6: The Group window used to view incoming e-mails and the Administrative Window under UMD window are packed.

The Pack/Unpack operations are primarily used to abandon a task for a while and open up space for other tasks. Packed windows can be restored to their previous sizes with a single Unpack operation. It is invoked by double-clicking on the packed window. Hence, Pack/Unpack operations allow fast task-switching and resumption.

Multiple Slide/Relocate Operations:

The Slide 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. It operation is accomplished by dragging with the middle button pressed.

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 7).

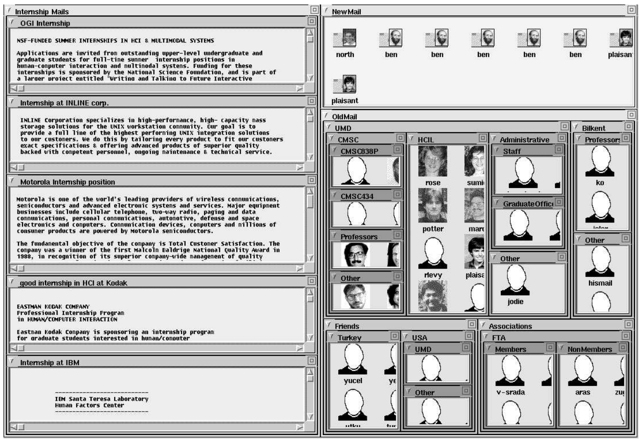

Figure 7: NewMail window is relocated to the top of the OldMail window.

Multiple Maximize Operation:

Users can focus on a set of windows and maximize them to cover the whole screen. This is particularly useful when users are expecting to work on a set of windows for a long time. With the maximize operation, users can use more of the screen estate, avoiding the loss of screen space due to nesting.

In Figure 8, the user maximizes the UMD window under OldMail group window.

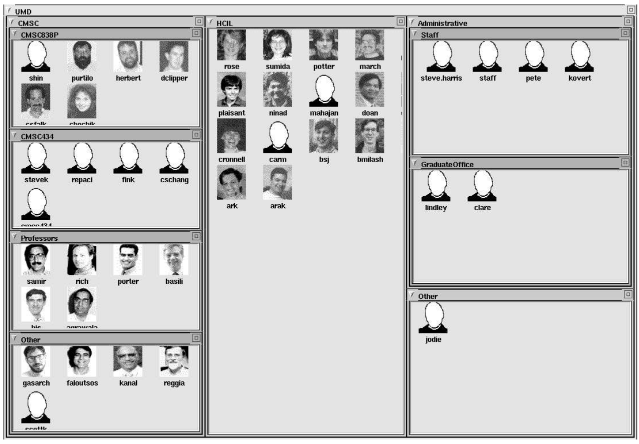

Figure 8: UMD window is maximized to work more easily.

Other operations:

Users can fix the width or height of a window in order to protect the window from unwanted resizing. This constraint on the window width or height can be removed by the appropriate release operation.

The layout can be saved and retrieved any time later to allow users to switch back and forth among different work layouts easily.

Users can undo or redo operations at any time. This makes the window operations reversible. Not all operations are undo-able e.g. close operation.

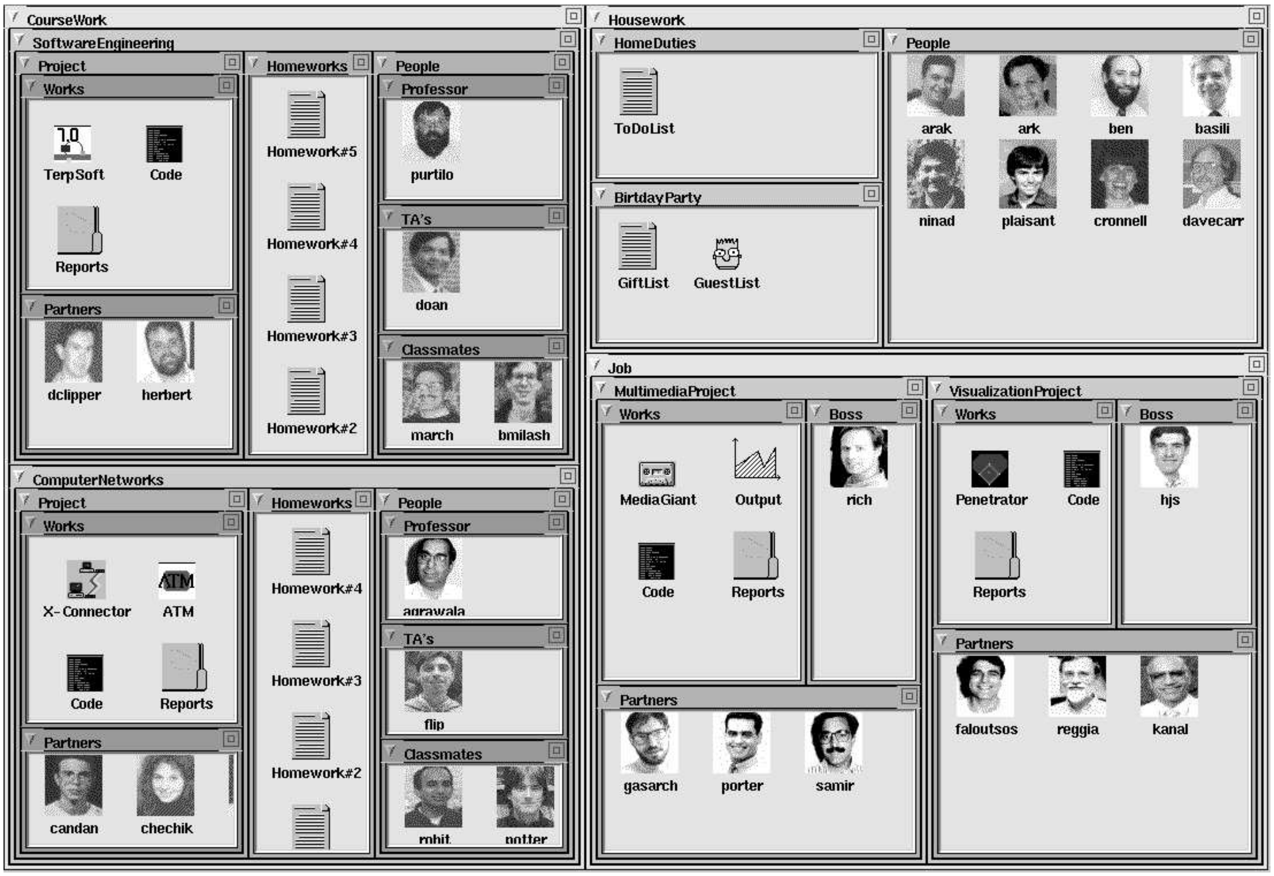

Figure 9: Hierarchical organization of different roles of a student: CourseWork window on the left holds all the information related to the two courses Software Engineering and Computer Networks that the student is enrolled in. Housework window contains windows related to the responsibilities at home, and Job window contains the two projects that the student is responsible at work.

Multiple window operations can also be achieved by serial application of a window operation to a number of windows. Once the users selects the operation, it can be applied to any window by clicking on the window. This operation is particularly useful when removing a number of windows from a group.

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 [22, 19]. 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.

Figure 9 shows an example mapping of different roles of a student onto a hierarchical window organization. This student takes two courses this semester: Software Engineering and Computer Networks. Project materials and partners, homeworks, and correspondence with the professor, TA's, and classmates are organized in a hierarchical fashion for each course. This student has a number of other roles like the organization of a birthday party, home duties, and job responsibilities. Partners, schedules, tools, and documents regarding each of these roles are mapped hierarchically into different windows. The layout clearly indicates the semantic relationship between the contents of the windows by the spatial cues in the organization of windows.

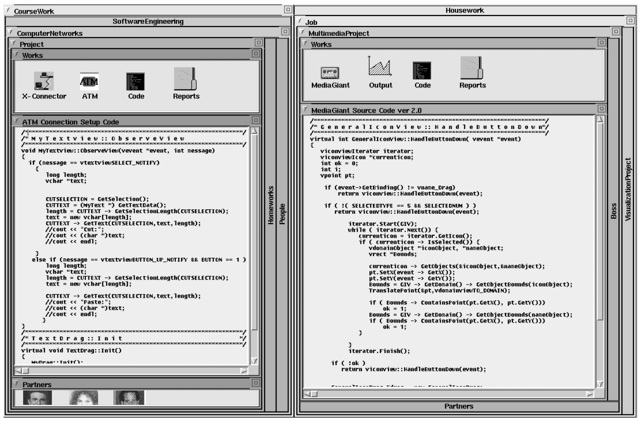

Figure 10: Layout customized to reference the code for Networks project at school, while working on the code for the Multimedia project at work.

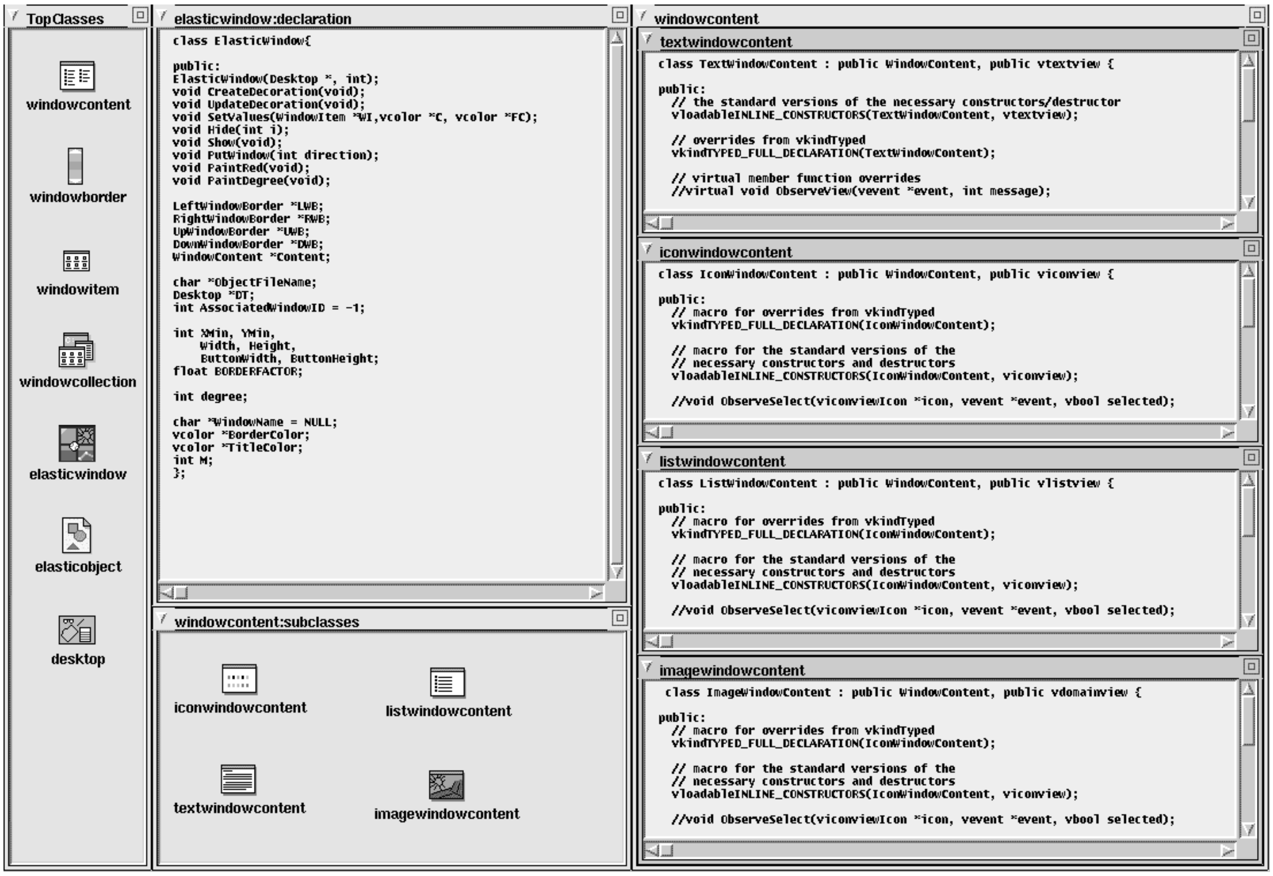

Figure 11: Programming Environment for object oriented programming: Class and subclass declarations organized in multiple windows

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

Table 1: Comparison of the number of window operations in the PRM example

The layout provides users with an overview of the roles, in which they can pick any task regarding a role easily and customize the layout for that task. Figure 10 shows an example customization of the layout to enable the student to reference the code of the Network project at school while working on the code for the Multimedia project at work. While working on the code, the user still has an overview of the roles since the packed windows keep the same spatial relationship. This layout can be considered as an example of detail within overview technique, where users are relieved from the burden of merging the overview and detail mentally. Comparison of the elastic windows approach with the independent overlapping windows approach shows the substantial reduction in the number of operations (Table 1).

With the use of multiple window operations, as well as resize, pack, and unpack 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.

Object Oriented Programing Environment:

Developing and maintaining large complex software systems is a difficult task. Generally many programmers work together on different pieces of the software and then integrate these pieces to build the system. In large software development projects where the applications, tools, code, and data are distributed across the network, programmers typically loose track of where the application code lies. Productivity will certainly increase when access to individual parts of the software components becomes easier and faster.

Most of the software development environments in use today are file-based. In these environments, programmers organize code as separate windows for each file. Programmers typically need to reference different parts of the code, such as data declarations, procedure declarations, bodies, and invocations, as well as related documentation like execution charts, and reports. It is beneficial to provide programmers with environments that enable them to group information and manage multiple windows easily on the desktop.

Object oriented technology is widely used in industry to overcome problems of large complex software development. Below we will describe possible benefits of an object oriented software development environment using elastic windows principles.

Figure 11 displays a view of a large project as seen by a programmer. On the left, top-level object classes are displayed as icons, where programmers can pick any of them to see their declarations, related documentation, or subclasses derived from them. The top middle window displays a class declaration, whereas the bottom middle window displays the subclasses derived from a base class. Icons representing these subclasses can further be selected to view declarations grouped in a single window as shown in the rightmost window.

As shown in Figure 12 documentation regarding a class can be viewed easily while updating the declarations for that class. To use more of the screen estate, windows or groups of windows can be packed as in Figure 12. Managing multiple windows at a time saves time when organizing windows to meet new screen space requirements of the current task.

Providing programmers capabilities to organize information and easy access to related information increases manageability of a complex system. This example also demonstrates the use of multi-window operations to reduce the burden of window management and the use of the Pack/Unpack operations to meet changing screen space needs.

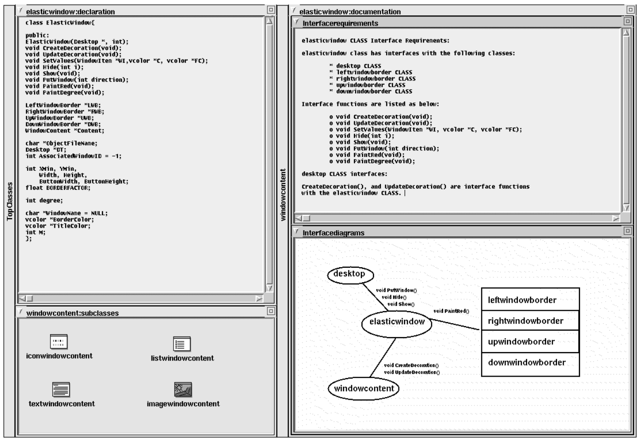

Figure 12: Window organization quickly changed to view documentation regarding a component of the system.

RELATED WORK

The Rooms system [13] 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.

The Cedar [24] system also uses tiling, where windows are organized in two columns of arbitrary number of windows. It also uses space-filling tiled layout, but proportional resizing is not provided. Windows can not be grouped hierarchically and multiple window operations are not provided. Windows minimized are iconized at the bottom of the desktop, possibly causing disorientation if the number of windows is high.

The Dylan programming environment uses a pane-based window system [10], which allows both horizontal and vertical panes, with a mechanism to create links between panes. The Dylan programming environment does not support multiple window operations and hierarchical organization of windows.

Xerox/Star [23], RTL/CRTL [8, 9], and Windows 1.0 also used tiling, but hierarchical organization and multiple operations were not provided. CIWM [11] uses automated window management. Although automatic strategies in window management relieve the burden of window management, direct user control is preferable as in most HCI artifacts. Myers has an excellent taxonomy of these early windowing systems [17].

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. We believe it is particularly useful for complex systems like CAD/CAM systems, and Object Oriented Software Development systems. Application of elastic window ideas in WWW Browsing seems promising by the introduction of textual objects which can be associated with windows or hierarchies of windows. 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 graceful animations of elastic window interactions.

The possibility of using overlapping windows to provide multi-window operations and providing a layout that enable users to group windows and apply operations on them is on our agenda of future research directions.

A video demo on elastic windows is available as part of the HCIL Open House'96 video report.
Contact hcil-info@cs.umd.edu for ordering information.
Refer to http://www.cs.umd.edu/projects/hcil/ for further information on the project.

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*, (Dec 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, (1983), pp. 54-57.

3. Bederson, B., B., Hollan, J., D., Pad++: A zooming graphical interface for exploring alternate interface physics, *Proc. of the UIST'94, User Interface Software and Technology Conference*, pp. 17-26.

4. 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.

5. 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.

6. 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.

7. 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.

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

9. 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.

10. Dumas, J., Parsons, P., Discovering the way programmers think about new programming environments, *Communications of the ACM*, (June 1995), 38, 6, pp. 45-56.

11. 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.

12. Gaylin, K., B., How are windows used ? Some notes on creating empirically-based windowing benchmark task, *Proc. CHI '86 Conference - Human Factors in Computing Systems*, ACM, New York, NY, (1986), pp. 96-100.

13. 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.

14. Kahn, M., J., Charnock, E., How to prevent "Window-its" in your Graphical Interface ?, *Proc. Silicon Valley Ergonomics Conference & Exposition, ErgoCon'95*, (1995), pp. 18-25.

15. 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.

16. 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.

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

18. 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.

19. 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), pp. 14-22.

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

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

22. Shneiderman, B., Plaisant, C., The future of graphic user interfaces: Personal role managers, *People and Computers IX*, Cambridge University Press, (Aug 1994), pp. 3-8.

23. Smith, D. C. et al., The Star user interface: An overview, *Proc. National Computer Conf.*, AFIPS Press, Arlington, VA, (1982), pp. 515-528.

24. Teitelman, W., A tour through CEDAR, *IEEE Software*, 1, 2, (Apr 1984), pp. 44-73.