WindowScape: Lessons Learned from a Task-Centric Window Manager

CRAIG TASHMAN and W. KEITH EDWARDS, Georgia Institute of Technology

People frequently experience difficulty switching between computer-mediated tasks. To help address this, we created WindowScape, a zooming window manager that uses implicit grouping to help users sort windows according to task. WindowScape was intended to provide a more flexible and intuitive grouping model than prior systems. We report on the design process leading up to the system, and alternative designs we explored. We describe a series of formative evaluations that resulted in significant modifications to our initial prototype, as well as a deployment study of the final version, where users lived with WindowScape on a day-to-day basis. Our results from this study reveal how users react to novel aspects of our system, including its particular uses of miniaturization and its approach to grouping. We also discuss the impact of a task-oriented approach to window management on other aspects of user behavior, and the implications of this for future system design.

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Windowing systems*

General Terms: Design, Human Factors

Additional Key Words and Phrases: Activity management, scaling, spatial organization, deployment evaluation

ACM Reference Format:
Tashman, C. and Edwards, W. K. 2012. WindowScape: Lessons learned from a task-centric window manager. ACM Trans. Comput.-Hum. Interact. 19, 1, Article 8 (March 2012), 33 pages.
DOI = 10.1145/2147783.2147791 http://doi.acm.org/10.1145/2147783.2147791

1. INTRODUCTION

Knowledge work often involves a range of activities, interleaved throughout the day. These interleaved users activities—which we refer to here as *tasks*—may include changing one's focus to a new project, scheduling a meeting, doing research on the Web, or responding to email. However, although we know that knowledge workers frequently switch between different tasks, actually doing so is often difficult [Czerwinski et al. 2004]. Not only do these transitions entail recalling the details of the task to be performed, but they can include the additional challenge of gathering the relevant work materials and arranging the workspace appropriately; in the digital world, for example, this work includes things such as rearranging windows, finding necessary files, opening and closing applications, and so forth. Not surprisingly then, computer users report significant difficulty in certain task transitions, such as those coming after

In the *Proceedings of the 2006 ACM Symposium on User Interface Software and Technology (UIST)*, the authors published a four-page Technote describing the initial prototype of WindowScape, which we cite as Tashman [2006] in the article.

Authors' addresses: C. Tashman (corresponding author) and W. K. Edwards, Georgia Institute of Technology, Atlanta, GA; email: craig@cc.gatech.edu.

DOI 10.1145/2147783.2147791 http://doi.acm.org/10.1145/2147783.2147791

an interruption, or where the task is of high complexity [Czerwinski et al. 2004]. Consequently, a wide array of systems have been developed to help users with this problem of task switching (a subset of what we might call *task management*, which may include not just task switching, but how tasks are "defined" in the first place, deleted, and so on), going back at least as far as 1986 [Henderson and Card 1986].

A number of systems have focused on better support for task transitions. For example, work by Bellotti et al. explored the difficulties of supporting multiple tasks within an email client [Bellotti et al. 2003]. However, one of the most common and ubiquitous challenges people face when switching among tasks is the need to rearrange on-screen resources; for example, retrieving now-relevant windows while perhaps hiding those related to the prior task. Thus, many have posited that task switching could be supported through window managers that allow windows to be manipulated in groups that correspond to these tasks [Henderson and Card 1986; Kandogan and Shneiderman 1997; Robertson 2000; Smith et al. 2003; Robertson et al. 2004]. However, despite the reported problems with task switching, and despite the large body of research that has focused on providing better task support, most of these alternative window managers have not caught on with users [Czerwinski et al. 2004]. Even Virtual Desktop Managers (VDMs) modeled after Rooms, which are included in some commercial offerings (such as Apple's MacOS), still appear to be used by only a minority of knowledge workers [Hutchings and Stasko 2004].

Several authors raise important concerns that may help to explain why these seemingly necessary tools have had such a poor reception. Robertson et al., for example, discuss the problems with the "strict separation" that VDMs and other Rooms-like systems place between tasks [Robertson et al. 2004]. Bardram et al. [2006] found negative reactions from users for having to manually specify their window arrangements, which these systems generally require. And in Tashman [2006] we similarly argued against the near universal assumption made by these window managers that each window belongs to only one task. In short, as we shall discuss in more detail, these systems' notions of what a task is, or how it is defined, seem less flexible than the ways in which users actually work.

In response to these weaknesses, we developed WindowScape, a task-oriented window manager. The system is designed support multitasking through more flexible ways of grouping windows, including allowing windows to exist in multiple groups and using an implicit approach to group creation. An initial prototype of this system was presented at UIST 2006 [Tashman 2006]. In this article, we build on this earlier work in three ways: (1) we elucidate the design process and trade-offs leading up to our initial prototype, including several alternative designs we considered and why we chose to develop the system as we did; (2) we discuss the current, final version of WindowScape, and explain how findings of several pilot studies helped bring it to its final form; and (3) we present the results of a deployment study of WindowScape where we explored the viability of the system itself and sought to understand users' reactions to it.

Evaluations of window management tools typically involve only laboratory studies, probably owing to the considerable difficulties of making such tools robust enough for deployment. Those systems that have been evaluated through longer-term deployment studies, such as GroupBar [Andrews et al. 2003] have typically relied only on very small numbers of participants, potentially making results difficult to generalize. CAAD was deployed to a larger number but focuses more on grouping files than windows [Rattenbury and Canny 2007]. Ringel also ran a study of how existing VDM users manage their desktops, but the focus was more on understanding how users behave than on the successes or failures of the existing systems [Ringel 2003]. Thus, for the most part, it remains unclear how well prior task-centered window management systems actually fit in, or adapt to, user work practices. Consequently,

we chose to evaluate WindowScape in situ by deploying it to thirteen users for a period of approximately ten days. Our findings help shed light on how users adopt and use task-oriented approaches to window management generally, as well as provide reflections on the specific design and features of the WindowScape system.

2. RELATED WORK

Even since the early days of personal computing research, there has been a desire to support interleaved user activities. Windowing was one of the earliest ways to address this issue, as it provided a way to divide the display space into multiple work areas. Early window managers divided the space through tiling, assigning a fixed region of the display to a given application or terminal session. Overlapped windows, first proposed by Alan Kay in 1969, initially appeared in SmallTalk in 1974 [Myers et al. 2000]. Since that development there have been occasional explorations of the benefits of overlapped versus tiled windows, with some research indicating increased efficiency of the latter [Bly and Rosenberg 1986].

But windows, on their own, can lead to messy, unmanageable work spaces, and numerous systems have been put forward to keep windows more organized, or otherwise make the mess manageable. One type of approach that has appeared often is scaling. Perhaps the most prominent example of this is Apple's Exposé, which temporarily scales down and tiles all open windows to let the user find one that was not readily visible.[1] Similar ideas are found in Windows Vista's Flip3D[2] and the Ring and Shift plugins for the Compiz compositing engine/window manager.[3] A different take on this idea is found in SCOTZ, which uses a tree map to scale windows differently depending on frequency of use [Tak and Cockburn 2010]. These systems, though, operate by scaling all windows concurrently to give the user an orderly but brief overview of her workspace. An alternate approach that has been explored sporadically over the course of more than two decades is to let the user selectively scale down and organize windows that are not immediately needed, reducing the number of full-sized windows and simplifying organization. The traditional approach to this is to iconify windows, replacing them with small, highly simplified proxies, typically representing only the type of content contained [Myers 1988]. Systems such as Sapphire let the icons change depending on the window content, and other systems, such as the x/uwm window manager, or Scalable Fabric, go further and show a scaled down thumbnail view of the window itself [Myers 1988; Robertson et al. 2004]. Apple's MacOS is similar in this regard but more structured, only allowing the user to arrange window thumbnails along the one-dimensional dock.[4] But while these and other uses of scaled windows have played an important role in the development of modern window managers, there has been little investigation into user reactions to manually arranged thumbnails as a way to represent windows. As discussed shortly, however, one objective of our research is to help provide additional insight into this through our evaluation of WindowScape.

Rather than scale down windows, an alternative approach has been to scale up the workspace. This is found in various "virtual screen" plugins to the GWM window manager,[5] and to some extent, in more modern window managers like Beryl.[6] Rather than scale down the windows, these systems provide a large virtual workspace that can be panned, such that the user may arrange windows over a continuous area much

[1]See http://www.apple.com/macosx/what-is-macosx/expose.html for details.
[2]See http://windows.microsoft.com/en-US/windows-vista/Using-Windows-Flip-3D
[3]See http://www.compiz.org/
[4]See http://support.apple.com/kb/HT2474?viewlocale=en_US
[5]See http://tronche.com/gui/x/gwm/html-manual/package.html
[6]See http://linux.softpedia.com/get/Multimedia/Graphics/Beryl-19790.shtml

larger than his monitor. In some cases, these systems then let the user zoom out to see an overview of his larger workspace, as with the "Anders Holst's Virtual Screen."[5]

And one of the most popular approaches to helping users manage their windows is based on the idea that certain groups of windows are more likely to be used together than others. Henderson and Card made this argument, suggesting that different user tasks may include different windows which are likely to be used together [Henderson and Card 1986]. They model this as analogous to hierarchical memory management: Just as certain pages of memory are more likely to be used together, the same is true with windows; people tend to use windows in groups. Thus, as with memory, one can typically increase efficiency by providing some mechanism privileging the windows that are members of the set with which the user is presently interacting, akin to a cache [Henderson and Card 1986].

Along these lines of reasoning, there have been frequent attempts to help users organize windows, or other content, into groups. One early example used a book-like metaphor for arranging projects into chapters [Feiner et al. 1982], while another provided hierarchical nesting of project spaces for Smalltalk [Goldberg 1984]. Rooms came shortly thereafter, providing a strongly metaphoric working environment in which rooms (i.e., workspaces) were connected by doors, and windows could be carried in one's pocket [Henderson and Card 1986]. In contrast to earlier systems, Rooms was based on statistical analyses of actual user behavior.

In spite of its insightful design and considerable influence, in the years since Rooms was created a variety of systems have sought to better support actual user work practices, cognition, or otherwise differentiate themselves. In order to understand these systems and their relationship to WindowScape, we will consider them from two perspectives: the first is the structuring of window groups: what a group consists of, how it relates to other groups, etc., and the second is group creation: how, and by whom, new groups are defined.

It is important to note that managing windows is just one of several ways to support users as they transition between tasks. Instead of operating on the window level, a variety of projects have sought to help users better manage files according to task (e.g., Rekimoto [1999]), and manage emails according to task as well (e.g., Gwizdka [2002] and Bellotti et al. [2003]). Nonetheless, since WindowScape operates firmly on the window level, we principally restrict our discussion to similarly window-oriented tools.

2.1. Window Group Structuring

As noted above, Henderson and Card's Rooms [Henderson and Card 1986] was among the first to employ the idea of virtual spaces, each of which could contain a different set of windows [Henderson and Card 1986]. Although the idea has been used heavily in the many modern VDMs, as well as in other forms in research projects such as the Task Gallery [Robertson 2000], Rooms' grouping model allowed a window to appear in multiple groups at once. Rooms supported this by separating the content of a window from its visual representation. The representation was contained in a "placement," which held the parameters such as the location and size of the window being used to display the content. This distinction enabled the user to copy a window's placement from one workspace into another, effectively letting the window be visible in both workspaces, while possessing a different size/position in each. In some cases, other systems have allowed this as well, such as the "dvrooms" GWM plugin.[7] But notably this functionality is absent from most projects of this sort, even those introduced after

[7]See http://tronche.com/gui/x/gwm/html-manual/package.html

the patent on the technology expired, including Apple's Spaces,[8] KPager,[9] or Beryl,[10] perhaps because of the additional conceptual overhead associated with representing overlapping groups to the user. The result, though, is that modern VDMs, and similar systems, effectively assume that a window pertains to exactly one workspace. Users must choose exactly one group in which to put a window, neglecting the possibility that a window might belong in several groups at once. Likewise, VDMs and similar systems generally require a window to be in at least one group. Yet even this seemingly benign requirement may be burdensome. The user may not yet have a clear enough conception of the properties of a window, or its relationship to his other windows, to determine which group it belongs to. This is all the more relevant if, as is commonly assumed, people seek to organize their windows according to activity; it remains very unclear when in a window's lifecycle users become cognizant of the larger activity of which a window is a part. And while some systems that employ a generally VDM-like model, such as the Activity-Based Computing client [Bardram et al. 2006] and, to a lesser extent, Rooms, do allow windows to exist outside of any group. This functionality is atypical among such systems.

An additional limitation of VDMs comes from the strict separation they impose between window groups. By placing the different groups of windows in isolated virtual spaces, these systems can help to hide irrelevant or undesirable content [Hutchings and Stasko 2004], but this comes at the cost of flexibility: it becomes difficult to fluidly access a collection of windows that happens to span several spaces. One must typically break their grouping scheme by moving the desired windows into the same space, performing the intended actions, and then returning the windows to their original locations.

In recognition of this limitation, several window grouping systems have been designed to facilitate access to windows across groups, such as Scalable Fabric, Kimura and GroupBar [MacIntyre et al. 2001; Smith et al. 2003; Robertson et al. 2004]. These systems support a variety of ways to flexibly organize windows, but ultimately still limit windows to occupying at most one group at a time. As with VDMs, this is a considerable limitation. For example, if users group windows by activity, they are left in a quandary of where to place something like an email client which naturally relates to multiple activities. Thus, as we describe shortly, one of the principle benefits of WindowScape's grouping model is that it supports window membership in multiple groups simultaneously, from both a metaphoric and functional level.

2.2. Window Group Creation

Among systems that seek to facilitate window access through grouping, there are several common approaches to creating groups, that is, conveying which items are associated with which others. Perhaps the most common is manual, in many cases prehoc, selection. That is, a given grouping must be explicitly, manually created, at least in a preliminary form, by the user before she can begin to take advantage of that particular group. This is the approach taken in Rooms, Task Gallery, Scalable Fabric, GroupBar and, to a large extent, the Activity-Based Computing client [Henderson and Card 1986; Robertson 2000; Smith et al. 2003; Robertson et al. 2004; Bardram et al. 2006]. This approach gives the user a high degree of control, but has disadvantages as well. First, it requires the user to define groups prior to the point in time at which she needs to access them, when their membership may not yet be clear. Second, it potentially requires the user interrupt her activity with the meta-activity of organizing windows; that is, even

[8]See http://support.apple.com/kb/ht1624
[9]See http://developer.kde.org/~larrosa/kpager.html
[10]See http://linux.softpedia.com/get/Multimedia/Graphics/Beryl-19790.shtml

if the user does know where a window belongs, the point of window creation may not be the best time to bother her with the task of articulating that. Recent work in activity management supports this, particularly when the user needs to make these decisions repeatedly [Bardram et al. 2006]. Of course, systems like Scalable Fabric or Group Bar do not require users to organize their windows, but until they do, they cannot fully benefit from the system's grouping functionality.

An alternate approach to identifying which items are associated with which groups is to attempt to infer the user's activities, and which objects belong to each, based on the user's past actions. Windows, or other items, can thus be grouped based on these inferred activity relationships. This approach is used in varying forms by UMEA, TaskTracer, and CAAD [Kaptelinin 2003; Dragunov et al. 2005; Rattenbury and Canny 2007]. UMEA, for example, observes the details of the users' actions, and attempts to create task groupings and assign items appropriately. Naturally though, no such system will infer perfectly; hence, work is left for the user in the cleanup of task contents [Kaptelinin 2003]. TaskTracer takes more of a mixed initiative approach, initially requiring users to indicate their current task, but with the intent that the system will gradually be able to infer the user's current task and thus assist in task transitions [Dragunov et al. 2005]. Both of these systems offer interesting possibilities for automatic task inference, but so far both also require a substantial amount of explicit user input.

A third approach enables the user to specify groupings based on their tasks, but less explicitly than with the systems described before. One example of this is Push-and-Pull Window Switching, which sees sets of windows that do not overlap as potentially being used together, and lets the user switch between them as groups [Xu and Casiez 2010]. An alternate way to implicitly group windows relies on the notion that user interface abstractions, like files and windows, often have multiple representational states, each corresponding to a different level of user interaction, what we might call a different level of prominence. Files, for example, can be viewed as icons in a file browser or be opened in applications; windows can be visible and interactive or iconified and thus hidden. We can consider this in light of the assumption that users will naturally bring into the more interactive, prominent state all of the objects they are working with together. When the user is ready to move onto some other activity it is further assumed that he will reduce the prominence of the current group of objects and increase the prominence of some other group. For example, one might switch activities by closing the documents needed for the previous activity, and opening the documents needed for the next. In so doing, the user has implicitly indicated which documents are used together: those that are open together. There is thus the opportunity to let the user select a group of documents with which to work by selecting a point on a timeline. Note that neither the system nor the user ever explicitly declares a particular grouping structure to be correct (i.e., which objects definitively belong to which groups). Rather, the system makes historical states reaccessible under the assumption that some of them will correspond to points in time when the user was performing one task or another, and that those represent the groupings of documents, windows, etc., to which the user will wish to return. The "objects" whose prominences are changed depends on the particular tool.

One example of this approach is Rekimoto's Time Machine Computing [Rekimoto 1999]. This system employed a timeline metaphor to enable the user to indicate which group of documents should be open by selecting a point in time. WindowScape employs this approach as well, but rather than applying it to files, applies it to windows. That is, as described in detail shortly, WindowScape allows the user to select points on a timeline to indicate which group of windows should be shown most prominently compared to other windows. As with other approaches to defining groups, the use of a

timeline carries challenges as well. Foremost among them is providing the user a way to refer to a desired point in time. Literal time-of-day, for example, seems inadequate. Rather, some sort of rich representation of the timeline must be provided to allow the user to find salient points that would correspond to the times of interest.

Although each of the outlined approaches has challenges, as described earlier we chose to focus principally on the use of timelines to support creating and switching between window groups in WindowScape. This is in part because of the granularity of window grouping which we seek to support. Naturally, window grouping can occur on a variety of scales: grouping together all windows related to writing a paper, for example, versus using one group for windows related to finding the paper's references, and another for those related to making the figures, and so on. The former, more encompassing notion of groups, is akin to Mark and Gonzalez's "activity spheres" [Mark et al. 2005]. These activity spheres may persist and be retrieved numerous times over a period of days or weeks. By contrast, Czerwinski and Horvitz consider finer-grained units of work, each described as being an average of 53 minutes in duration [Czerwinski et al. 2004]. The fact that work can be decomposed into activities on such widely varying timescales suggests the opportunity to tailor an intervention to a particular granularity of grouping. WindowScape was therefore intended to support shorter-term units of work, like those described by Czerwinski et al. [2004]. This choice of granularity is a key motivator towards our use of a timeline approach to managing groupings. Specifically, we assume that groupings of short duration require a commensurately fast, lightweight means of creating and switching between groups. We see timelines as offering this, since they provide the potential for window group transitions that do not require forethought about group composition. That is, users do not need to decide what windows properly belong together until they actually need to return to a group. By contrast, most window grouping systems, such as VDMs, require an advance decision. We also felt that representing window groups as points on a timeline would provide a flexible task model that would more easily accommodate the association of individual windows with multiple groups.

We see this use of a timeline for window group management as a primary differentiating factor of WindowScape. Though systems such as Time Machine Computing employ a timeline for task management as well, it is for solving a different problem, more centered on file management than window management [Rekimoto 1999]. The Kimura system is likewise related, but also focuses less on general window management than on workspace management with special large, high-resolution displays [MacIntyre et al. 2001].

3. WINDOWSCAPE SYSTEM OVERVIEW

We designed WindowScape with the broad goal of helping users to organize and retrieve their windows. Thus as with many of the aforesaid systems (e.g., GroupBar [Smith et al. 2003] and Scalable Fabric [Robertson et al. 2004]), the facilities for dividing windows into groups act as one part of an integrated collection of functionality for supporting window management. Here we provide a brief overview of WindowScape's general window management functionality, followed by its support for window grouping.

WindowScape starts with a zooming window manager, akin to that of Scalable Fabric [Robertson et al. 2004]. The user's windows are presented in miniature, thumbnail form, and may arrange as desired on the desktop (Figure 1(c)). These "miniatures" can be selectively expanded to full, interactive size either individually or in aggregate, and arranged and used as desired (Figure 1(d)). Expanded windows may then be miniaturized, again individually or in aggregate, returning to their prior miniature locations. To help users access occluded windows or miniatures,

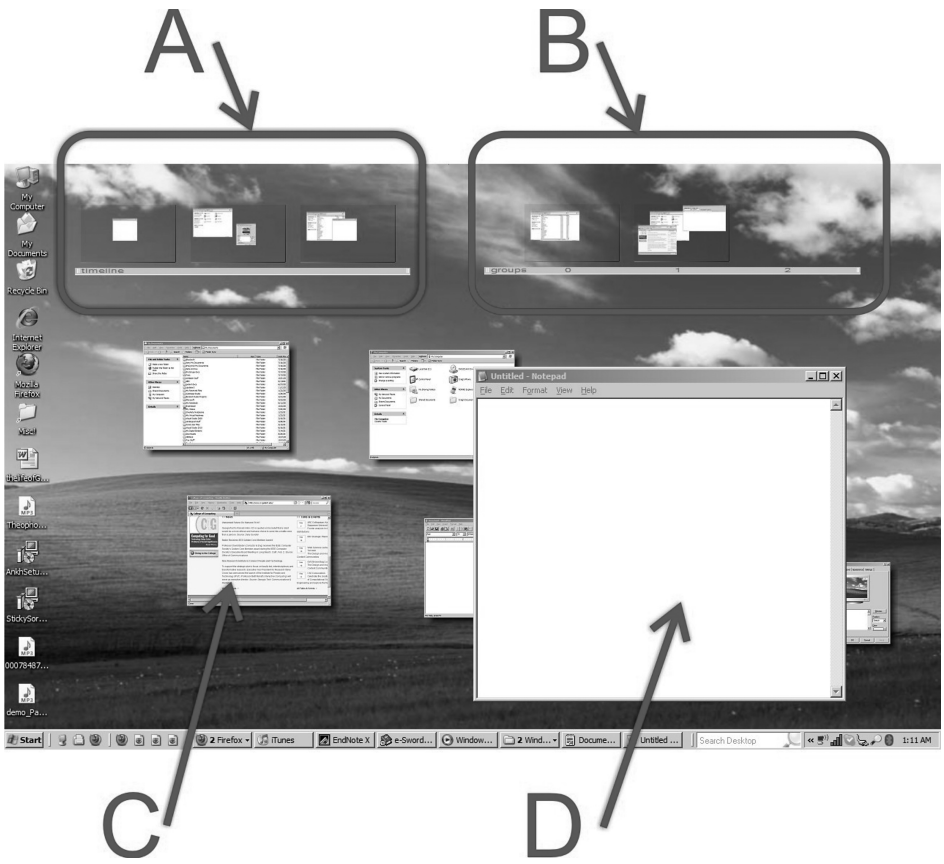

Fig. 1. Final version of WindowScape: (A) the timeline bar; (B) the bookmark bar; (C) a miniaturized (thumbnail) window; and (D) an expanded window.

WindowScape provides a collection of mouse and keyboard interactions for bringing all miniatures, and expanded windows' title bars, in front of other windows (Figure 2).

As discussed before, WindowScape enables the user to switch among groups of windows via a temporal interface, the timeline bar (Figure 1(a)). This presents a collection of snapshots, each depicting the windows that were expanded at a given point in time. Selecting a snapshot will change the expansion state, positions, and sizes of the user's windows so that they reflect the selected snapshot. The particular points in time shown on the timeline bar are determined through inference, as described shortly. However, users can also create snapshots explicitly, when they anticipate that they will want to return to a given state. These explicitly created snapshots are added to the bookmark bar (Figure 1(b)) and can both be created and accessed via several mouse or keyboard interactions.

In the following sections we begin by discussing our general design process and explain how we arrived at the larger window and grouping models used in WindowScape. Subsequently, we consider and explain the details of the system's user interface.

4. WINDOWSCAPE OVERALL DESIGN APPROACH

Because WindowScape's window grouping functionality is one part of its larger collection of support for window management, we began the design of the system by exploring

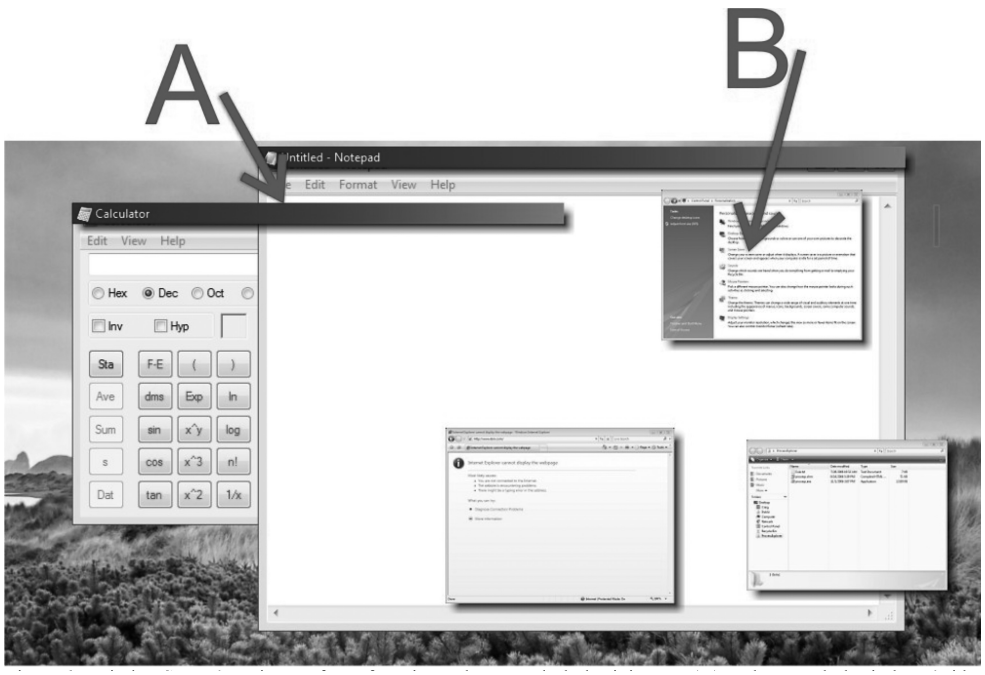

Fig. 2. WindowScape's bring-to-front function, where occluded miniatures (B) and expanded windows' title bars (A) are brought in front of the expanded windows.

several general approaches to managing windows, and in the context of the approach we chose, designed a grouping mechanism.

Throughout this section, we will describe our design process in general, and then discuss the rationale behind our approaches to window management and grouping, elucidating some of the trade-offs we encountered along the way. In the subsequent section, we will discuss how we embodied these broad approaches in the details of WindowScape's design.

4.1. Design Process

We began the WindowScape design process by exploring general window management functionality. We developed a variety of prototype systems, sometimes building on well-established techniques such as VDMs, or otherwise developing more novel interactions. In several cases, we prototyped these systems and used them internally to refine our designs and develop a stronger intuition for how different window management functionality can be used in practice.

As our designs gradually led us to the form WindowScape ultimately took, we constructed a prototype version of the system running on Windows XP. Our choice of operating system was motivated in part by our interest in potentially conducting a deployment of the system, and Windows XP was, and to a lesser extent still is, the most common desktop OS. This would also ensure that our system would be compatible with subsequent Windows versions with minimal modifications. The resulting prototype was discussed in Tashman [2006].

As we continued to develop the system, we conducted a series of small pilot studies with the aim of identifying: (1) basic usability and learnability problems with our design, and (2) how best to evaluate the software and understand whether and how

people might incorporate it into their actual task management practices. As these were pilot studies, they explored several different types of evaluation methods, rather than striving for scientific consistency or generalizability. As such, we began with several laboratory evaluations assessing task completion times using WindowScape versus the traditional Windows XP shell. As discussed shortly, these studies helped identify several opportunities to improve the efficiency of WindowScape's interactions, but also revealed that a laboratory study was likely not the appropriate means of assessment. Such a study would not give participants adequate time to learn and adapt to WindowScape to provide for a fair controlled study, nor would it measure longer-term behavioral changes or reactions. We thus concluded our pilots with a three-user deployment. This predeployment study helped identify several important usability issues relating to regular, day-to-day use that were not obvious from the pilot lab studies.

In response to these pilot tests, we substantially revised WindowScape to better fit users' requirements. This included closer integration with the host operating system, broadening its compatibility to include Windows Vista, and changing several of the interaction techniques. So, to observe how users reacted to WindowScape and to assess the viability of our specific interactions, we concluded with a deployment study giving the software to thirteen participants for a period of about one week. We sought to triangulate on user reactions through a combination of interviews, system logs, and questionnaires, and report on our results shortly. First, though, we explain WindowScape's approach to window management and to grouping.

4.2. Window Management Approach

In selecting a window management approach for WindowScape, we sought to satisfy several goals. First, we wanted to support a rich variety of cues for helping users identify their windows, including: (1) using the general appearance of a given window, and (2) the spatial location of the window, which is known to significantly aid in retrieval [Tak et al. 2009]. We also sought to provide users with peripheral awareness of their window environment, as open windows are sometimes used as cues to remind users of earlier tasks [Hutchings and Stasko 2004], and this would also facilitate faster window retrieval if users can already see the windows that are available.

In some cases, though, we found these goals to be in tension. For example, supporting window identification by appearance, if nothing else, requires fairly large images for the user to perform accurate recognition. But the larger these images are, the less freedom we can offer the user in spatially organizing them such that they do not significantly overlap and hinder the very recognition we are trying to support. Likewise, even if we allow for some overlap, assuming the user will compensate by rearrangement, this effectively forces the user to regularly alter his spatial layout, denying him stable spatial cues that would help him in retrieval. The goal of maintaining peripheral awareness raised a similar tension: providing the user a continuously visible layout of his windows, especially if those windows are large enough to recognize easily, and the space they inhabit is large enough that they can be arranged freely, could be expected to consume a great deal of the user's display space. As a result, the user may have little room left to work.

While we found no "silver bullet" solution to satisfying these opposing criteria, we considered several possible approaches to see which provided the best balance.

One potentially obvious option is to use the standard Windows Taskbar or MacOS Dock, both of which offer persistent overviews of one's windows, and employ thumbnails in certain cases to support window recognition. Neither of these, however, offers the user much opportunity to control the spatial layout of her windows.

We likewise considered "shading," an interaction[11] common in Linux where, in this case, all windows could be made to collapse such that only their title bars remain visible. This approach supports gaining an overview and flexible spatial positioning, but does not support recognition of windows by appearance.

As discussed earlier, Exposé, found in MacOS, offers a mode the user invokes to scale down and tile all windows (or all windows belonging to a given application), allowing the user to see these windows in parallel. After a window is selected in this view, the windows return to their original positions, with the selected window in the foreground. We considered something in this vein which does a laudable job of supporting window recognition by appearance. However, we were concerned that it gives the user virtually no control over where windows will appear in the tiled view. Exposé also couples a window's position in the tiled view to its position in the nontiled view, diminishing the consistency of the tiled view, and hence the extent to which users can rely on spatial memory in retrieval.

Finally, we also considered a scaling window manager similar to what is found in Scalable Fabric [Robertson et al. 2004]. In this approach, windows can be toggled between full size and thumbnail; the user expands windows as they are required, and shrinks them down again when they are not immediately needed. The thumbnails are typically left behind expanded windows, but can be brought to the front when required. They can also be arranged as desired, with their positions independent of the windows' full-size positions, that is, the user can change the full-size position of a window without changing its thumbnail position. This provides a spatially stable area in which the user can arrange the thumbnails for her windows, supporting recognition by appearance and by spatial location. And since the thumbnails are always visible in the background, they provide a partial peripheral cue of the user's other windows. Ultimately, these advantages led us to take this type of approach with WindowScape; nonetheless, it does also entail several trade-offs that are worth noting. Most obviously, unless the user explicitly brings the window thumbnails to the foreground, the peripheral cues they provide are only visible to the extent that full-size (i.e., not thumbnail) windows are not occupying the user's display. Likewise, there is still a trade-off between the sizes (and hence recognizability) of the thumbnails and the extent to which they can be flexibly rearranged. But as discussed next, we sought to include functionality to at least partially mitigate these issues.

4.3. Grouping Approach

In designing a grouping method, a central goal was to minimize the burden on users entailed by taking advantage of groups. Since, as discussed before, users may not initially know what window grouping scheme they require, we sought to let them defer their decisions about groupings (i.e., which window belongs to which group) as long as reasonably possible, that is, until they actually need to reaccess that group. Similarly, when the user does have to decide what windows belong to a group, we sought to place a minimum of constraints on that decision, and thus required that a window can belong to many or even zero groups at a time.

In light of these requirements, we chose a basic grouping approach that we describe as *tacit* grouping. Here, the user never actually specifies the group to which a window belongs. Instead, the user is presented with an assortment of different candidate window groups, such as in a list, and simply chooses the one she would like to access. This approach has several advantages. Foremost, it lets the user defer any explicit judgments about which windows ought to be used together until the very moment they

[11]See http://sawmill.sourceforge.net/prog-manual.html#Shading%20Windows for an example.

are needed; so if the user does not initially know which windows belong together, that decision may be postponed until the moment the user needs to return to that group. This also avoids pigeon-holing windows into a single group, since the same window can naturally belong to different groups in different candidate group configurations.

But in contrast to more traditional grouping approaches, tacit grouping has several notable challenges as well. First, groups are not represented as concrete, persistent objects that the user can arrange, position, and remember; rather, they become merely options in a selection the user makes from time to time. The result is that, in contrast to their physical, real-world counterparts, groups become ephemeral, persisting only while one is using them. A consequence of this is that users can not necessarily really rely on things like spatial memory when retrieving a group, since the user never creates/positions the group in the first place. This leads to the second challenge, that while this approach defers the question of grouping, it potentially imposes a large burden on the user at that moment of returning to a window group, as the user may have to search through a daunting list of candidate groupings to find the one desired. As a result, this approach to grouping hinges on the means by which the system determines, and displays, which of the multitudinous possible groups the user may actually want to access.

Our first strategy for providing this was machine learning prediction. To begin, we recorded all the sets of windows that were full size at the same time. We then set up the scaling window manager (above) so that the user could select a collection of thumbnails before expanding them all at once to their full-sized scales and positions. Thus, each time the user selected a thumbnail, a WindowScape would use the user's window expansion history to predict what other thumbnails the user wanted expanded at the same time, and would automatically highlight them in the WindowScape UI. If the user agreed with the prediction, she could press a hotkey and the thumbnails she selected, as well as the thumbnails that the learning algorithm predicted, would all expand simultaneously. If the prediction was wrong, the user could continue selecting thumbnails, and after each selection, WindowScape would continue to refine its prediction.

Ultimately though, we found this approach to identifying candidate window groups to have several limitations. First, depending on the first window that a user selects, the accuracy of the predictions can vary widely. For example, if the user first selects a window that happens to be frequently used in three distinct window groups, the prediction algorithm likely has only a 1/3 chance of guessing correctly. Though this could likely be improved marginally by factoring in the transition order between groups, this approach ultimately may still put too much of a burden on the user to select the right thumbnail(s) first for the predictions to be useful. The second, related problem came to light through our internal testing of the system, that we rarely sought to expand more than two or three windows at a time. As a result, if we sought to expand three windows, then if WindowScape had not offered a correct prediction after the first selection, then generally, the effort required to select the second thumbnail, assess the prediction, and engage the hotkey was comparable to simply selecting the second and third thumbnails. Thus, the prediction often provided too little benefit. As a result, we required an approach that would effectively offer a candidate set of predicted window groups before the user even selected any thumbnails, something that would offer a prediction before the user gave any indication of which windows he wanted next.

In light of this requirement, we ultimately selected a timeline visualization, in which the user is presented with a listing of recently visited groups (i.e., sets of windows that were all full sized at the same time), and simply chooses the group to which she would like to return. Effectively both this and the machine learning approach leverage the user's history to try to suggest a group to revisit. But whereas the prior approach used

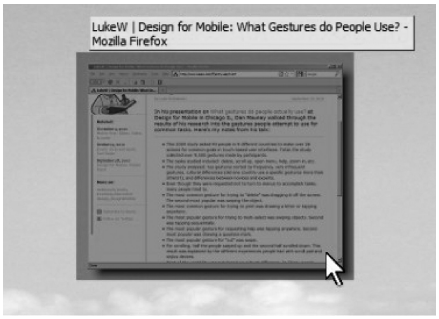

Fig. 3. Mouse hovering over a miniature, which highlights the miniature and displays a text label.

the user's partially expressed intention to identify the best candidate in the search space, this approach transfers that burden to a recognition task for the user. That is, we are shifting the core of the user's burden from expression to recognition. The viability of this approach is thus dependent in large part on the scale of the search task, however, we suspected this search task to be manageable for two reasons: First, preliminary indications from work like GroupBar suggest that people only wish to switch between a small number (about 2.5) of groups at a time, suggesting that the user's search task would be quite tractable [Smith et al. 2003]. Second, even if the numbers of groups were greater, we hypothesized that most group switches would be to other recently accessed groups, potentially because of users switching repeatedly between original tasks and interrupting tasks [Czerwinski et al. 2004]. This is a kin to the theory underlying Rooms, namely that windows can be divided into sets such that users are more likely to switch within a set than between sets. We hypothesized the same to be true of groups of windows, so that even if there were many candidate groups, the most recent groups would usually be the ones the user will want to revisit. In the next section, we discuss the specific user interface we developed for this timeline system, as well as the other functionality in WindowScape.

5. WINDOWSCAPE SYSTEM IN DETAIL

While our early design research helped define the overall approaches we took to WindowScape's window management and grouping, our subsequent prototypes and pilot studies helped to develop these basic designs into concrete, specific user interfaces. In this section, we explain the details of WindowScape's user interface and provide the basis for some of our design choices.

5.1. Window Miniaturization

In this section, we consider WindowScape's general window management functions. We explain in detail the behavior of the window miniaturization operations and the support for finding occluded windows, and we provide background for how some of these functions arrived at their final form.

5.1.1. Miniature Window Appearance. WindowScape displays miniature windows as $\frac{1}{4}$ size ($1/16^{\text{th}}$ area) thumbnails, but with added shadowing to ensure they can be differentiated from the background in spite of their scaled down borders. In our early prototypes, WindowScape did not show window titles to maintain a simple, clean appearance. Our pilot studies, however, indicated that users would have benefitted from additional miniature recognition cues; thus we added labels showing the miniature's title, but only while the user is hovering over a miniature (Figure 3). We felt this was a reasonable trade-off between simplicity and information.

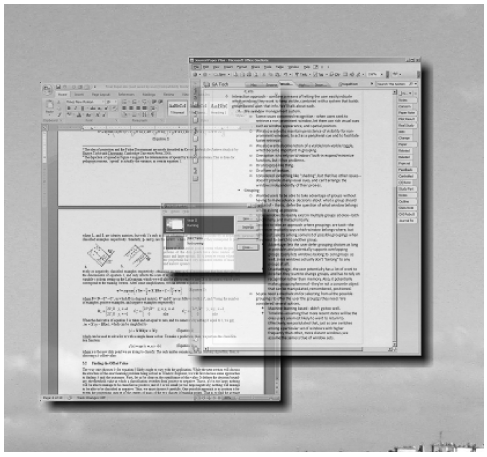

Fig. 4. Three overlapped miniatures (right) are shown translucent, so the user can see the occlusion.

5.1.2. Positioning and Occlusion. A key principle in miniature window positioning was to preserve the layout the user creates. Thus, miniatures may be dragged to form an arbitrary arrangement by the user, and they will remain in those locations indefinitely, and will return to these positions when miniaturized regardless of how they are positioned when the user re-expands them. This approach to positioning also informed our handling of miniature overlap. One natural way to handle overlapping objects is to slightly reposition nearby objects to ensure all are ultimately visible (as in Robertson et al. [1998, 2004]). A possible disadvantage of this approach, however, is that, while the changes may be small, moving an object could alter the positions of other objects the user positioned with care. Rather, we chose to make overlapping miniatures transparent (Figure 4), which ensures spatial stability is preserved. And though it does not guarantee all miniatures are immediately reachable, it does provide the user a cue when a miniature is occluded, alerting her to the presence of the covered object(s).

5.1.3. Expanding and Miniaturizing Windows. From the earliest prototype of WindowScape, expanding a miniature window was performed simply by clicking it with the mouse. We explored more options, however, for expanding multiple windows in parallel. Normally, expanded windows stay above miniature windows in the z-order. As such, we considered two ways to expand multiple miniatures: (1) select multiple miniatures then perform an "expand-all" operation, and (2) invoke a special mode that keeps the miniatures on top of the expanded windows. The former method was included in early prototypes of WindowScape, and allowed the user to right-click miniatures to highlight them without actually expanding them. The user could then left-click any of the highlighted miniatures to expand them all. While this was in many ways adequate, in some scenarios we expect that the user would need to see the content of one window before he can determine which other windows should be expanded as well. Thus, in our final system, we also allow the user to hold down the Shift key while left-clicking miniatures, which expands the miniature window but keeps all other miniature windows in the foreground, thus allowing the user to select additional miniatures. Releasing the Shift key returns the miniature layout to the background (Figure 5).

In addition to WindowScape's built-in interactions for expanding windows, we sought to provide a cohesive user experience by integrating with several native Windows shell functions that are used to access windows. The first is the Windows taskbar, in which the user can select the button corresponding to a window to bring it to

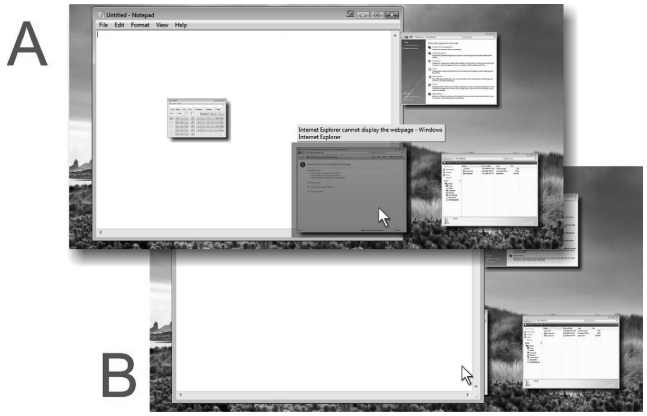

Fig. 5. Holding down the Shift key while expanding miniatures (A) keeps the miniatures visible over the expanded windows. Releasing Shift brings the expanded windows back to the foreground (B).

the foreground. The second is the window recency list invoked by Windows' Alt-Tab keyboard command. When the user attempts to retrieve a miniaturized window with either of these functions, WindowScape will expand the window in addition to giving it the keyboard focus.

Once windows are expanded, we likewise sought to provide a rich set of interactions for miniaturization. For aggregate miniaturization, we offer two options: The first is simply to double-click any empty area of the windows desktop background (i.e., an area of the desktop background that has neither icons nor miniature windows) which miniaturizes all expanded windows. The second option is a keyboard shortcut,[12] Ctrl +/, which performs the same function.

In developing support for miniaturizing individual windows, we observed that this functionality would in some ways be similar to Windows' native "minimize" function, which hides a window, leaving it accessible via the taskbar, as both of these reduce a window's prominence. Since Windows already provides a rich set of features for minimizing (with pre-existing shortcut keys, and a minimize button on each window's title bar), our first question was whether to co-opt the minimize function and remap it at a low level to WindowScape's miniaturize command, as was the case in Scalable Fabric [Robertson et al. 2004]. But in spite of the advantages of not forcing the user to become familiar with an additional set of keyboard shortcuts and UI components, we opted to separate minimize from miniaturize for two reasons: First, we felt it unwise to remove any functionality native to the Windows shell, as many users likely developed habits around the behavior of the native minimize function. The second reason is that minimize is potentially a useful function in its own right, providing a way to further conceal infrequently needed windows than WindowScape's miniaturize command. As such, we provided two ways to miniaturize a single window: the first is a keyboard shortcut, Alt-Esc by default. The second is to click a button which WindowScape adds to the title bars of all windows, alongside the standard Windows shell title bar buttons (Figure 6(c)).

5.1.4. Accessing Occluded Items. With miniature windows lying behind expanded windows, one of our requirements in designing WindowScape was ensuring that the

[12]Most keyboard shortcuts in WindowScape can be changed by the user, so the shortcuts we note are just the defaults.

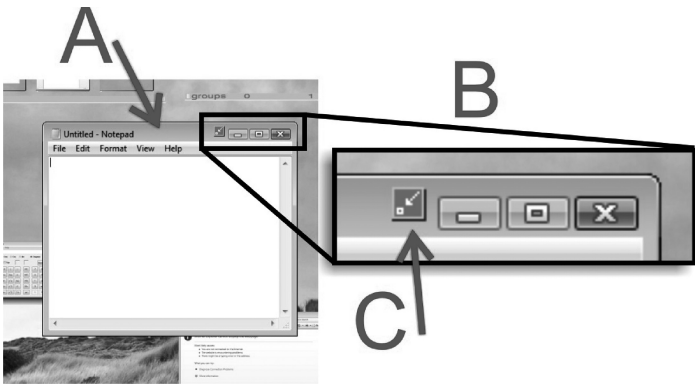

Fig. 6. A callout (B) of the title bar (A) of a window, showing WindowScape's miniaturize button (C).

user could easily access occluded miniature windows while maintaining a generally stable screen image.

Scalable Fabric provides an interaction in this vein, allowing the user to move the mouse to the edge of the screen to bring all miniature windows to the foreground, and then do the same thing to send them to the background again. Czerwinski et al., however, found that the average number of documents in a task ranges from 1.6 to 2.5 [Czerwinski et al. 2004], so we suspected that when users already have one window expanded, it is rare when they will need to expand more than one additional window at a time. Thus, we sought to optimize around the scenario of adding one more window, while still supporting the addition of more. For this, we provided two interactions, one mouse based, and one keyboard based.

The mouse-based interaction is simply to drag the mouse (i.e., press and hold the mouse button while moving the mouse) over the windows desktop background, which brings all miniatures to the foreground (Figure 2(b)). When the user releases the mouse all miniatures return to the background; and if the user releases the mouse over a miniature, it will be expanded. As a result, in a single mouse movement, the user can view his miniature layout and expand an additional window. We realized that we could further generalize this interaction by additionally bringing the title bars of expanded windows to the foreground (Figure 2(a)), allowing the user to easily access occluded expanded as well as miniaturized windows.

5.1.5. Keyboard Navigation. As noted earlier, WindowScape provides a keyboard shortcut for bringing all miniature windows (and title bars of expanded windows) to the foreground. By default, the user simply presses the Insert key to invoke this mode, and releases it to exit this mode and send the miniatures back to the background. And while the hotkey is pressed, the user may press the arrow key buttons to select a miniature or title bar; on releasing the hotkey, the selected miniature will be expanded or title bar's associated window brought to the foreground.

Navigating around an arbitrary collection of objects by arrow keys presents challenges, though. Typically, keyboard navigation through a collection of objects starts with one selected object, and uses the keyboard's four arrow keys to switch to the closest object in a given direction. But proceeding blindly in the direction dictated by the arrow keys can easily leave some objects unreachable, if the desired object is not in a position that corresponds to the direction of one of the arrow keys (Figure 7(a)). To avoid this, many past systems set up the arrow key mappings so as to ensure all objects are reachable. The standard Windows shell does this in keyboard navigation of its icons; and Scalable Fabric does this in the keyboard navigation of its thumbnail

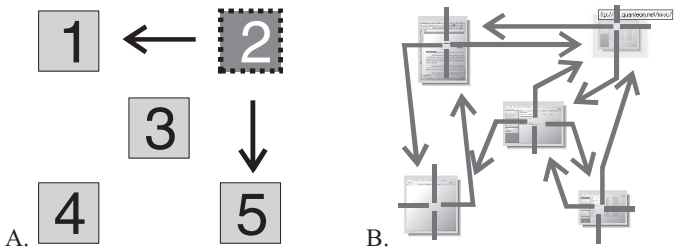

Fig. 7. A: Starting with object 2 selected, blindly following the directions of the arrow keys in navigation will leave object 3 unreachable; B: Guaranteeing reachability leaves transitions that are hard to predict.

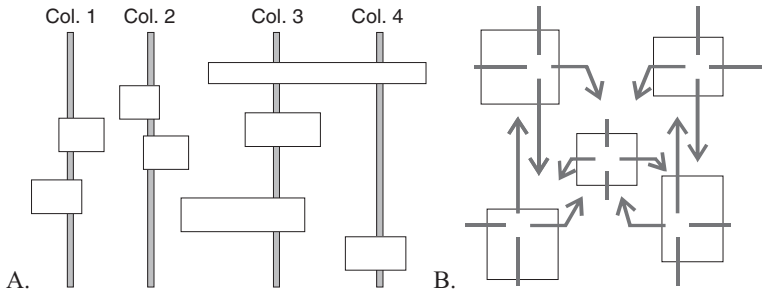

Fig. 8. A: Example of how WindowScape groups objects into columns as part of its keyboard navigation algorithm; B: Arrow key mappings for the type of situation depicted in Figure 7.

windows. The disadvantage of this technique, though, is that it can make navigation unpredictable, since it is difficult to know in advance what transition will be associated with any arrow key; see Figure 7(b) for an example taken from Scalable Fabric, where the red arrows show the transitions between each thumbnail for each of the four arrow keys. Note how pressing the right arrow key while at the bottom-left object selects the top-left object.

In designing the arrow key mappings for WindowScape, we needed to guarantee reachability, and thus we chose to focus on making the transitions more predictable. We sought to do this by choosing a transition mapping algorithm that appeared conceptually simple, and exposing it to the user. Thus, our mapping algorithm begins by sorting all objects into columns. It does this by starting with the leftmost object M1 and proceeding right, sorting into the same column any objects that overlap with M1. As the algorithm proceeds right, when it encounters the right-edge of any object in the current column, it considers that column finished and begins a new column, and continues along the same lines. Afterward, if it finds an object overlaps with two columns, the object is included in both. The result is columnar groupings, as in Figure 8(a). To navigate among the miniatures and title bars, the left/right arrow keys change the column selected, selecting either the next column to the left or right, respectively. The up/down arrow keys select the next object within the current column, above or below the selected object. As a result, the transitions among the five objects in Figure 7 would be as shown in Figure 8(b).

Essential to this algorithm is exposing the column groupings to the user in some way, so she can plan which arrow keys need to be pressed. In our initial prototype, we did this by showing which items were in which column groups, rendering a thick, dotted, vertical line behind each group of items. But as our pilot study indicated this could be distracting to users, our final version displays only which items are part of the currently selected, active, column group. As shown in Figure 9, this is displayed by

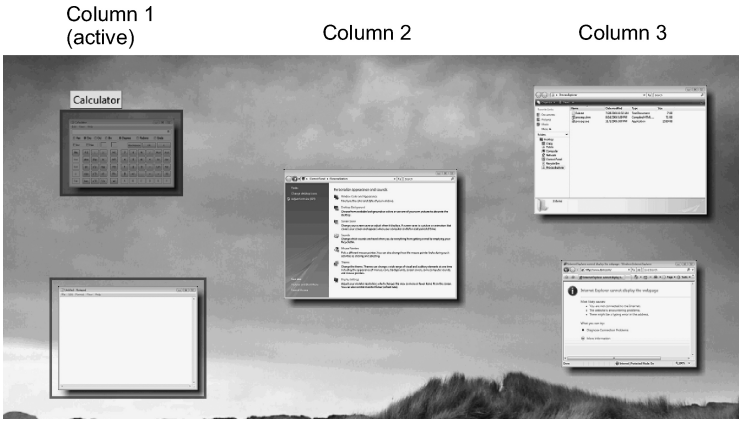

Fig. 9. In the final version of WindowScape, columns are depicted less explicitly. As seen in column 1, objects in the active column just receive a border to differentiate them from objects in inactive columns.

drawing blue rectangles around the items in the active group. The currently selected item within the active group is shown with a complete blue overlay (Figure 9).

5.1.6. Differences from Prototype. While the most significant differences between the final version of WindowScape and the prototype described in Tashman [2006] concern the grouping functions, we summarize several of the other key differences as well. First, the final version is far more closely integrated into the Windows shell. The earlier prototype, for example, occluded the native Windows desktop, whereas the final system integrates into it (as in Figure 1), and differentiates input meant for the Windows desktop from input meant for WindowScape. We also added text labels when hovering over miniature windows and allow the user to hold shift while expanding miniatures to keep them in the foreground. Finally, we changed the appearance of the keyboard navigation cues to create much less visual distraction.

5.2. Window Groups

As discussed earlier, WindowScape's basic approach to grouping is to use a timeline to let the user return to groups of windows that were earlier used together. In this section, we discuss the design and function of this timeline, and how it evolved from the earlier prototype version. And in addition to these implicit groupings, we also discuss WindowScape's support for more explicitly created window groups.

5.2.1. Timeline Design. In order to help users return to the windows they were using at an earlier time, WindowScape's timeline needed to offer users a visual representation of those earlier times. Here we chose to use a "photograph" metaphor, a small depiction (which we refer to as a "snapshot") of the user's display with the windows arranged as they were when that group of windows was last used (Figure 10). There are several trade-offs to this approach. For example, we hypothesized (and later verified, as discussed shortly) that window groups would sometimes correspond to the user's tasks. This is why we chose to have the windows in the snapshot depicted in the arrangement as when they were last used, to allow the image of the arrangements in the snapshot to act as an additional cue to the task corresponding to that snapshot. But the disadvantage of this approach is that it results in the images of some individual windows being extremely small, or overlapped by one another. Thus, to mitigate this, we display a listing of the included windows when the mouse hovers over a snapshot (as in Figure 10).

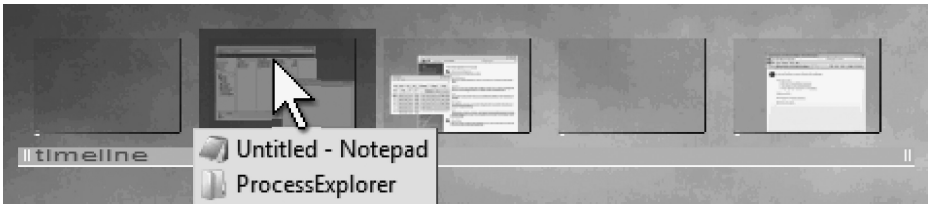

Fig. 10. The WindowScape timeline (shown also in Figure 1(a)) showing several "snapshots," each representing a different group of windows. The leftmost snapshot is the most recent.

A similar question centered on the images of the windows in the snapshot, as they could either show: (1) the way the window looked when it was last used with the other windows in that snapshot, or (2) the way it looks presently. The first option appears to be more amenable to supporting memory, as it would keep familiar the appearance of the overall scene depicted in the snapshot. But it is also deceptive, suggesting it can return the user to a given window with the depicted contents when it cannot do so. Thus we opted for the second option, sacrificing possibly better recognizability for a more honest interface.

For interaction with the timeline, the user simply clicks a snapshot to cause all expanded windows to miniaturize, followed by causing all windows depicted in the snapshot to expand and return to their depicted positions. Additionally, the timeline can be resized to control how many snapshots are shown.

5.2.2. Timeline Composition. In populating the timeline, we found a central tension between displaying many snapshots (and giving the user a variety of choices for groups to reaccess) versus displaying few snapshots (and simplifying the visual search task). In balancing these alternatives, our snapshot selection algorithm seeks to identify substantial changes to the set of which windows are expanded, attempting to filter out intermediate states. For example, if a user expands three windows in rapid succession, we want to see that as only a single state change. It thus identifies changes as substantial under the following conditions:

—if the user has expanded or miniaturized at least one window, and
—the user remained in the previous state at least 15 seconds, or
—at least one window was miniaturized, or
—at least two windows were expanded.

Though these heuristics are the result of trial and error, our internal testing suggested that they captured states at a reasonable granularity.

In our earlier WindowScape prototype, snapshots corresponding to each of these states were added to the timeline. But our internal testing suggested this very literal chronology resulted in an unnecessarily crowded list. As such, our final version culls redundant snapshots, for example, if two snapshots represent the same group of windows, only the most recent one is shown.

5.2.3. Keyboard Access to Timeline. To facilitate access to the timeline via keyboard, in our earlier WindowScape prototype, we simply made the snapshots "objects" that could be navigated using the same keyboard shortcuts described earlier in Section 5.1.5. But our pilot testing suggested this was inefficient, requiring the user to access the miniature layout and then use the arrow keys just to get to the timeline area. As such, we made a distinct mechanism for navigating the timeline by keyboard, and modeled it on the existing window recency list in Windows, the Alt-Tab keyboard shortcut. This keyboard shortcut brings up a list of recently visited windows; continued pressing of

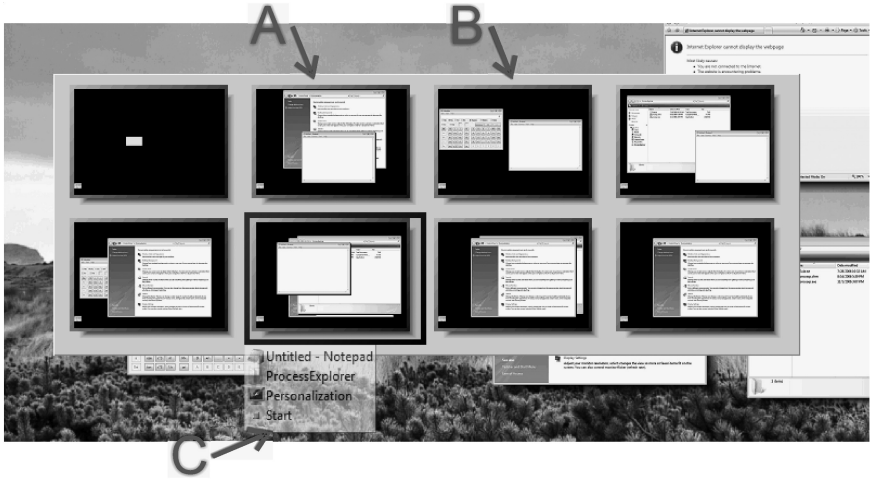

Fig. 11. The keyboard interface to the timeline.

the Tab key moves the user through the list, and releasing the Alt key brings the corresponding window to the foreground. Advanced users, including some in our pilot studies, are very familiar with this shortcut, and can use it to rapidly switch between windows.

WindowScape's keyboard-based timeline navigation is modeled on this interaction which, we expected, would be familiar to many of our users. Thus, when the user presses a shortcut key (the default is holding the Windows key while pressing A), a dialog box similar to that of Alt-Tab is brought up (Figure 11). When it first appears, the most recent snapshot is selected (Figure 11(a)) and the user can press the A key to cycle sequentially through the different snapshots (such as Figure 11(b)). When the user releases the Windows key, the selected snapshot is invoked, and windows are miniaturized, expanded, and rearranged so as to reflect the snapshot. Thus, as easily as the Windows shell lets users switch between windows using the keyboard, WindowScape lets users switch between entire groups of windows.

5.2.4. Explicit Groups. While WindowScape's implicit, timeline-based groups have various advantages in reducing the forethought burden on the user, they are also in some ways unstable. Infrequently used snapshots, for example, will pass off the end of the timeline. And even snapshots that are regularly accessed will still frequently change their positions on the timeline. Consequently, we sought to complement the implicit grouping with explicit grouping, in which the user can intentionally create a group with specific windows at a specific location. In doing this, we employed the same approach to representing a group as in the explicit case because it naturally allows windows to exist in multiple groups, and will save the user from having to learn an additional conceptual model of grouping. Thus, explicitly created groups are displayed in very much the same way as implicit (Figure 12, showing the explicit group list, which we refer to as the "Bookmark bar").

But while our earlier prototype let the user add snapshots to the bookmark bar by copying them from the timeline, we found that neither in our pilot studies nor our internal testing was this feature used. Thus, for the final system, we replaced this interaction with ways to add snapshots to the bookmark bar targeting two scenarios: (1) the user wants to add a snapshot for the windows currently being used, and (2) the user anticipates a group of windows will become important later and wants to add a

Fig. 12. Several snapshots in the explicit group list.

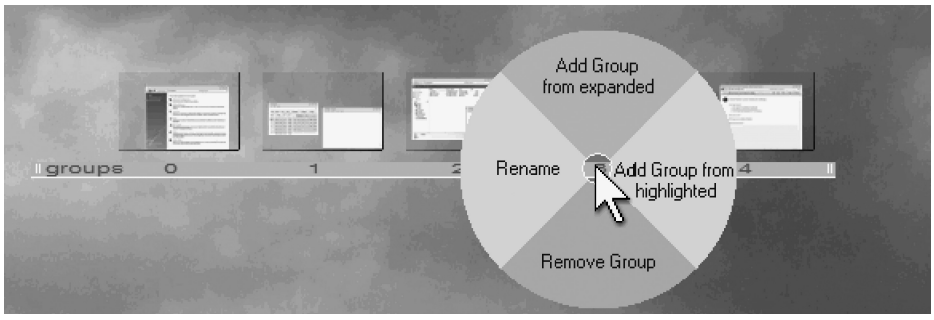

Fig. 13. The pie menu used to add snapshots to the bookmark bar.

snapshot in advance. We discuss the mouse and keyboard interactions for doing so in turn.

5.2.5. Creating Explicit Groups via the Mouse. To add a new snapshot to the bookmark bar, the user clicks on the numbered space where the snapshot is to be added, revealing a pie menu (Figure 13). Here, the user is presented options corresponding to the preceding two scenarios: Addressing the first, the user can add a snapshot consisting of the windows that are currently expanded. Addressing the second scenario, the user can also add a snapshot consisting of any miniature windows that have been highlighted; as noted before, miniatures can be highlighted by right-clicking on them.

5.2.6. Creating Explicit Groups via Keyboard. Focusing on the first of the previous two scenarios, we were inspired by certain development environments that allow the user to create or access a bookmark in one's code using similar shortcut keys. Thus, we allow the user to add a snapshot on the bookmark bar corresponding to the current set of windows, and to place this snapshot in a given numbered space, by pressing Win-Alt-[number]. For example, the user might add a snapshot of the currently expanded windows in the 4^{th} numbered space by pressing Win-Alt-4. To later return to this group, the user presses Win-4.

5.2.7. Differences from Prototype. As discussed earlier, besides various small changes in the labeling and visual design, the grouping functionality in the final version of WindowScape included several key differences from the prototype version discussed in Tashman [2006]. For the timeline-based grouping, our final version includes the distinct keyboard-based timeline interface for more efficient group switching, as well as a more parsimonious algorithm for deciding which window states to include on the timeline. In the explicit grouping functionality, we removed the concept of copying a snapshot from the timeline to the bookmark bar. Instead, we replaced it with functions for creating groups from the currently expanded windows or from highlighted miniatures.

5.3. Implementation

WindowScape was developed in C++ using the Borland C++Builder compiler, and interfaces with the OS mainly through direct Win32 API calls. Since we sought to support Windows XP as well as newer versions of the OS, we use the GDI and GDI+ APIs for WindowScape's graphics.

The majority of WindowScape's core functionality centers on window miniaturization, which we found surprisingly easy to support. Somewhat simplified, miniaturization consists of moving a window far off screen and replacing it with a new, smaller window that contains a thumbnail image of the original. The replacement window and its thumbnail start out at nearly the same size as the original window and are rapidly animated down to the final smaller size. To ensure smooth animation on Windows XP (which generally does not use off-screen buffering), we suppress normal window repaint messages until the final frame of the animation.

Some of the most significant challenges associated with developing WindowScape were in the integration with the Windows shell. Of course, we may have been able to avert some of these challenges if we targeted Linux, which readily supports replacing the window manager, especially by building WindowScape on the Metisse metawindow manager [Chapuis and Roussel 2005]. But because we sought to make a deployment study as feasible as possible, we opted to maintain compatibility with the popular Windows OS to maximize the number of people who could potentially use WindowScape. Likewise, even some advanced GUI frameworks for Windows were unavailable in this situation; Windows Presentation Foundation, for example, would have made the graphics trivial but then raised considerable challenges in interfacing directly with the Win32 Windows API as required to achieve full integration with the shell. Ultimately then, we necessarily chose to built WindowScape directly on top of the Win32 API.

Technically, WindowScape's integration with the Windows shell was accomplished through Windows hooks, a supported mechanism for intercepting certain events that occur within other processes. In WindowScape, we use this hook support to monitor basic shell events, including when a window is created or destroyed, so that we can create/destroy the required miniature windows or other data structures. We also use hooks to monitor the user's interaction with the desktop background, and the use of the Alt-Tab keyboard shortcut, as both can act as commands to WindowScape Finally, we also use hooks to add the miniaturize button to each window's title bar, by detecting when the window's title bar is being repainted or clicked. Cumulatively, WindowScape thus relies on a significant collection of hooks, but in practice neither we nor our study participants reported a noticeable drop in performance as a result.

5.4. Methodological Implications

While we designed WindowScape to support deployments, hence developing it for Windows XP, the value of this became most salient after our pilot studies. Throughout these preliminary evaluations, it became clear that while a laboratory study could yield general feedback on feature usability, as well as low-level performance data, such lab studies were not the ideal way to evaluate the notions of task-based window management more broadly. Primarily, this was because factors such as task completion time are perhaps not the most salient or useful factors that should be evaluated in a tool such as this. We were more interested in whether task-oriented features, such as those provided by WindowScape, were useful when trying to complete real work; similarly, we were interested in how users would appropriate these features into their day-to-day work habits, and how their work habits might change as a result. Additionally, as we learned during our pilot controlled studies, users often did not have the time to learn how to use our system effectively. For example, we found that users easily lost track of

some of WindowScape's features during our studies. One user even commented that his vast experience with the standard Windows shell was putting WindowScape at a disadvantage. Thus, we undertook a longer-term deployment study to mitigate the learning and novelty effects that might dominate a lab-based study, and to provide insight into how such task-oriented features impact users' real work, on their real tasks.

6. EVALUATION

In order to evaluate WindowScape in real, day-to-day use, we deployed it to thirteen volunteers, allowing them to use it for about ten days before debriefing them with a questionnaire and an interview.

6.1. Method

As WindowScape provides functions well beyond, but dependent upon, the basic window management functions in Windows, we recruited a group of thirteen skilled Windows users for our study. Participants received the software and user guide via email, and most also watched a short demo as part of our recruiting efforts. No compensation was provided, but participants were allowed to keep their copy of the software. WindowScape was installed to run at startup, and participants were asked to use it for 10 days performing the same work they otherwise would.

To gather structured, subjective feedback on the value and role of WindowScape in participants' work practice, we administered a Likert-style questionnaire at the end of the study. The questionnaire addressed several issues, including participant task management behavior before and after receiving WindowScape, the overall usability and learnability of the software, and the value of different WindowScape functions.

To solicit greater elaboration and discussion than was possible in the questionnaire, we also conducted semistructured interviews with participants, allowing them to provide deeper feedback on the areas that were most important to them. The interviews lasted about 18 minutes on average, and were attended by 12 of the 13 participants, as one participant did not come to the interview.

WindowScape additionally gathered log data on how and when participants used different features of the software, which participants sent to us at the end of the study. However, due to an incompatibility in our logging software and some participants declining to send us the log file, we received usage log data from 9 of the 13 users.

In order to analyze the data we used several statistical techniques, including bivariate Spearman correlation and Mann-Whitney tests, to identify relationships among survey questions or log results. For the interview transcripts, we performed a data-driven analysis, mainly with the intention of expanding on the results of the questionnaires.

6.2. Results

Our first goal in evaluating WindowScape was to understand which features people found useful. Figure 14 shows generally how participants judged WindowScape's main operations according to the questionnaire responses. When we ranked these by median response, we found miniaturization to have been considered the most useful, followed by the features for bringing occluded miniatures and title bars to the foreground. The data from the logs lent credence to this, showing that, of WindowScape's core commands, users most frequently used the miniature window expansion functions, followed by using the timeline and bookmark operations, and finally the "bring to foreground" (Friedman Two-Way ANOVA ranks were 2.44, 2.17, and 1.39 respectively, $p = 0.068$).

6.2.1. Uses. We included several questions in our survey to help us understand what types of benefits users felt WindowScape provided (Figure 15). We found that 9 of our

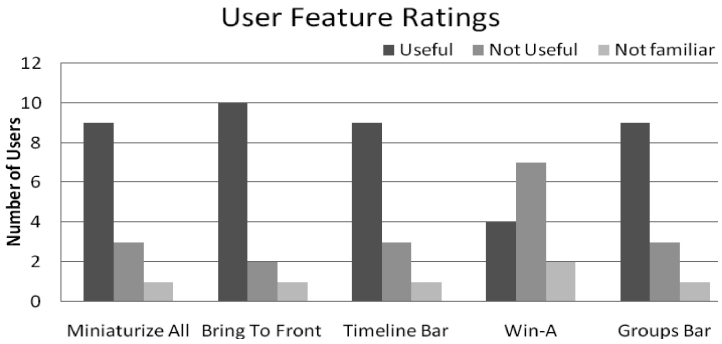

Fig. 14. Users who found WindowScape's features useful, not useful, and who were not familiar with the feature in question.

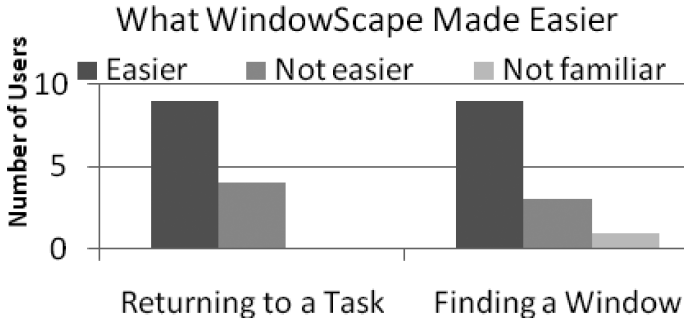

Fig. 15. Number of users who found WindowScape made task resumption and window finding easier, not easier, and who were not familiar with the type of functionalty in question.

13 users, or 69%, felt it made task resumption and window finding easier than in the standard Windows environment. However, users found it to provide more of a benefit for window finding than interruption recovery. Because WindowScape's window grouping functions were designed to aid in task switching, we expected users who found the grouping operations useful would also find WindowScape to be more helpful for task transitions. A correlation matrix of survey responses suggested this to be the case; users who found the bookmarks, in particular, to be more useful also rated WindowScape as more helpful in aiding task resumption (Spearman $r^2 = 0.81$, $p < 0.001$). Similarly, users who found WindowScape to be more helpful for window finding also tended to miniaturize windows, and reposition the miniatures, more frequently (Spearman $r^2 = 0.65$, $p = 0.015$; and $r^2 = 0.71$, $p = 0.008$, for miniaturizing and dragging, respectively). We discuss WindowScape's use in window management and task switching in more detail next.

6.2.2. Window Management Use. As noted before, 69% of our user sample found that WindowScape made it easier to find windows. Our interviews helped us to understand why. We noted that users typically attributed the improvement to WindowScape's miniaturization features. Two participants pointed out that the use of the window thumbnails allowed them to more easily distinguish their windows, and two other users alluded to that idea as well. Of the latter, one said in reference to the miniaturized layout, "*. . . you have all of them, so that you understand what window is what.*" A user also commented that being able to spatially organize the miniatures made them easier to find. Another user compared WindowScape to Alt-Tab, arguing that WindowScape

was superior since it does not require one to progress linearly through a list to reach one's windows. One of our questions in designing WindowScape was whether people could recognize windows by 1/16th area thumbnails (the scale used in WindowScape). We found that most did not report having had any problems doing so. This is in keeping with the results of others, including Robertson et al. [1998], who observed through controlled studies that people could successfully identify Web pages by thumbnail and spatial location. Our observation goes even further, suggesting that the types of content which individuals typically use in their real work are amenable to identification as thumbnails. However, two users did cite some difficulty, particularly in the case of similar-looking windows such as multiple instances of the same word processor application, as would be expected.

In addition to the miniaturization, users often cited the functions for bringing miniatures and title bars to the foreground as being useful, but typically in conjunction with the miniature layout, that is, bringing the miniatures to the foreground. However, one user who rated the "bring to foreground" features "extremely useful" did not even like the miniaturization, preferring instead to find windows by their title bars (as in Figure 2). Together, this supports our questionnaire responses, which showed these functions to have been the most useful parts of WindowScape. In general, participants made more frequent use of the window management functions than those for window grouping per se.

To better understand how participants used miniatures, we asked what strategy they chose in arranging them. The responses varied widely. One user put all miniatures on a secondary monitor, seemingly creating separate focus and context regions reminiscent of Scalable Fabric [Robertson et al. 2004]. Some reported positioning miniatures by relationship, or to maintain constraints like preventing overlap with other miniatures or desktop icons. Several users reported positioning them arbitrarily, and at least one user positioned them by priority (with more important miniatures higher up).

6.2.3. Window Grouping Use. With one of the goals of WindowScape being to develop a more flexible approach to representing window groups, we were interested in whether our snapshot metaphor would be clear and useful to participants. From our interviews and questionnaires, we found that all of our participants did appear to understand the metaphor, with 11 of the 13 finding either the timeline or bookmark grouping systems to be of some use. Of those, 5 described the bookmarks or timeline as "very" or "extremely" useful. The two users who found neither grouping mechanism useful seemed to attribute this to idiosyncratic issues rather than a lack of understanding the snapshots. The first user's computer had to be rebooted frequently, causing any groupings to be lost; the second was reliant on a VDM and said he would need more time to integrate WindowScape into his work habits.

We were also interested in the user response to our use of a timeline as a window grouping mechanism. Through several interview questions relating to confusion the user may have experienced, and to the usefulness of different parts of WindowScape, we found that 8 or 9 of our 13 participants[13] appeared to understand the basic purpose of the timeline, and how one would use it for switching between window groups. We found that the timeline bar displayed on the desktop tended to be more positively received than the Alt-Tab-like keyboard shortcut (see Figure 11), but we suspect this may be because the shortcut, likely being easier to forget, may require additional time to gain user adoption. Descriptions of how participants used the timeline were in line with our expectations. One participant, who used WindowScape's bookmarks as his main means of window grouping, described the timeline as useful for when he "lost track

[13]One person's answers were unclear.

of windows." A second user described switching groups with the Alt-Tab-like timeline interface, which he said he tended to use while having larger numbers of windows, between ten and twelve, open at a time.

Some users however, had more difficulty using the timeline. As noted, four or five[13] of the 13 were not clear about its purpose, or how it was intended to be used. Some of these felt we should have shown scenarios to better explain it. One felt he did not have a "system" for incorporating it into his existing work practices. Of those who found it to be useful, two also felt more predictability was needed regarding when new snapshots would be added to the timeline.

The second mechanism that we provided to support window grouping, the bookmarks, was well received, with users often preferring it to the timeline. As we expected, users who liked the bookmarks used them for grouping related windows together. One participant noted how he would create groups for different projects. Others described grouping windows more by category than by project. One explained how he would group his development environment with a Web browser window used to lookup function references, while using another group for tasks related to emailing. Another participant similarly described putting his development windows in one group, and media windows in another.

For accessing the bookmark groups, participants generally used the bookmark bar shown in Figure 14. Most were unaware of the bookmark keyboard shortcuts, but several who found them felt they were quite useful. Of the six users who were familiar with the keyboard shortcuts for adding and retrieving bookmarks, they gave the features median ratings of "very" and "somewhat" useful, respectively. One user who rated them very highly described them as similar to controls from a video game he had played.

Of the 9 users who found the bookmarks to be useful in some capacity, we found that 6 were not using a VDM or other window grouping tool before installing WindowScape. We took this as an indication that experienced computer users can often find useful applications of windows grouping quite quickly, even if they are not accustomed to doing so. Similarly, we found 2 of the 4 participants with VDMs rated the bookmarks as "very useful," while the other 2 rated them as "not useful at all" and "somewhat useful."

The users did point out some areas where we could improve the bookmarks. Some people had difficulty understanding the interface, which appeared to stem from our use of pie menus rather than more conventional Windows controls. At least two participants also wanted bookmarks to encapsulate program states so they could survive a reboot, which WindowScape does not currently support.

Although many users found the window grouping functions useful, we found the sizes of the groups to be quite small. The log data showed that when a user selected a snapshot (either from the timeline or bookmarks), it contained an average of 1.9 windows (S.D. across users of 0.7). Though some users occasionally accessed larger groups, none contained more than 4 windows.

6.2.4. Task and Window Awareness. Through the surveys, users reported median responses of 3 and 4 on Likert-scale questions (1 to 5, 5 is "Strongly agree") asking how much more aware users felt of their tasks and windows, respectively, when using WindowScape. We discussed this with users, and found that most of the increased sense of window awareness came from the general window management functions, like the miniaturization, and being able to bring miniatures to the foreground (Figure 2). Specifically, we found several participants said that they used the miniatures to maintain awareness of open windows that were not central to their current task, and that the miniature layout lent itself to being "glanced" at. One participant said, "You can get it [a window] out of the way, but still be aware of it." When asked why

Pre-WindowScape Computer Use	
Question	***Median Response***
Have projects involving 2+ windows	"Almost all"
Have windows pertai ning to 2+ projects	"Constantly"
Use maximized windows	"Almost never"
Have open windows from multiple projects	"Constantly"
Change proj ects by:	(where 5 is very often; 1 is very rarely)
First closin g windows of prior project	2
Put windows for new project on top of windows from prior project	4
Minimize windows from prior project to taskbar	4
Put windows for new project on different monitor	2
Put windows for new project on different part of screen from those of prior proj ect	3
Other	N/A
Number of windows open at once	6-10
Use of Alt-Tab	"Several times per hour."
Using WindowScape	
Impact of WS on your computer use	"It made using my computer somewhat better"
Usefulness of miniaturizing all windows	"Very useful"
Usefulness of brin ging miniatures to foregrou nd	"Somewhat useful"
Usefulness of timeline on desktop	"Slightly useful"
Usefulness of Alt-Tab-like timeline shortcut	"Not useful at all"
Frequency of using timeline	"Almost never"
Usefulness of bookmark bar on desktop	"Somewhat useful"
Usefulness of bookmark keyboard shortcut	"Not familiar with function"
Frequency of using book marks	"Several times per week"
Usefulness of WS for ret urning to past project	"WS made it slightly easier"
Usefulness of WS for find ing windows	"WS made it somewhat easier"
"WindowScape fit well with the way I work."	3 (Where 5 is strongly agree)
"Using WindowScape, I felt more aw are of my projects that I was working on."	3 (Where 5 is strongly agree)
"WindowScape helped me feel more aware of which windows I had open."	4 (Where 5 is strongly agree)
Computer use with WindowScape	
Projects with 2+ windows	"Far more than half"
Had windows related to 2+ projects open at once.	"Several times per day"
Had 1 window related to 2+ projects.	"Several times per day"
Had windows maximized	"Several times per day"
Keep open more/fewer windows then before WS	"The same number"
How many windows open at once?	Between 3-5 and 6-10
How often is Alt-Tab used?	"Several times per week"
Likelihood of keeping WS	4 (Where 5 is very likely)

Fig. 16. Median responses to select survey questions.

the miniaturization was her favorite feature, another said, "... because it was giving me an overview of whatever activities were being done." Surprisingly, one user who did not even find the miniaturization useful appeared to enjoy a peripheral awareness of her email client, referring to it as a sort of "security blanket."

The relationship between awareness and the use of miniaturization was further supported by the use logs. We found a significant correlation between how frequently users interacted (e.g., expanded, repositioned, etc.) with miniatures, and the amount of increased awareness of their windows they reported (Spearman $r^2 = 0.65$, $p = 0.029$). Likewise, we found a similar relationship to window awareness with the reported usefulness of the miniaturization functions ($r^2 = 0.70$, $p = 0.003$).

6.2.5. WindowScape's Impact on Users. In our survey, we gave participants several questions asking them to estimate any changes in their behavior resulting from the use of WindowScape. We expected that participants would keep more windows open at once with our tool; and 5 participants reported this to be the case. One specifically pointed out how, instead of using more tabs in his Web browser, he was opening more browser windows. We also found participants reported significantly less use of Windows' Alt-Tab while running WindowScape (Wilcoxon signed rank test, $p = 0.012$), which we attribute to our tool allowing users to find windows more easily. Two participants supported this claim by comparing Alt-Tab to WindowScape with a preference for the latter, particularly when performing multiple tasks in parallel.

6.2.6. General User Impressions. When surveyed, we found most participants liked WindowScape. We asked if they planned to continue to use our tool after the end of the study, and on a 1 to 5 Likert scale (5 is "very likely") the median response was 4. We sought more subjective feedback as well, enquiring during the interviews how well WindowScape fit users' work habits. We found that 7 of the 12 users who were interviewed felt it fit well with their existing habits, with one who felt his work habits instead adapted in light of the tool (e.g, by keeping more windows open). Several users also noted they found WindowScape "cool" to use. For example, one specifically liked the miniaturization for its entertainment value, finding himself "obsessed with it!" Another echoed this idea, saying that WindowScape was, "cool to look at; it was fun to show people."

Finally, we asked users whether they felt a tool like WindowScape was necessary, or whether the status quo was adequate. We found 9 of the 12 interviewed felt more than the status quo was needed, at least in some circumstances. In particular, three of the users said that the standard Windows environment is sufficient for smaller numbers of windows, but that a tool like WindowScape is needed as the number grows. To note, one user put that threshold at about 6 windows, while another put it at 10 to 12. In describing the need for better window management, one user said the increasing use of tabs in web browsers serves as evidence of the problem, while another felt something like WindowScape should come integrated into the operating system by default.

7. DISCUSSION

Beyond helping us to identify the more and less effective aspects of the WindowScape system itself, the results of our deployment study suggest a variety of implications for the design of future window and task management systems. These come in the form of identifying the results of specific design strategies that we implemented in WindowScape, but also the behavior of users as they coped with their task and window management requirements. In this section we discuss several of these implications as well as consider the significance of the results for our design of WindowScape.

As discussed before, prior systems have exemplified a variety of approaches for the creation of groupings. However, very few window grouping systems have supported

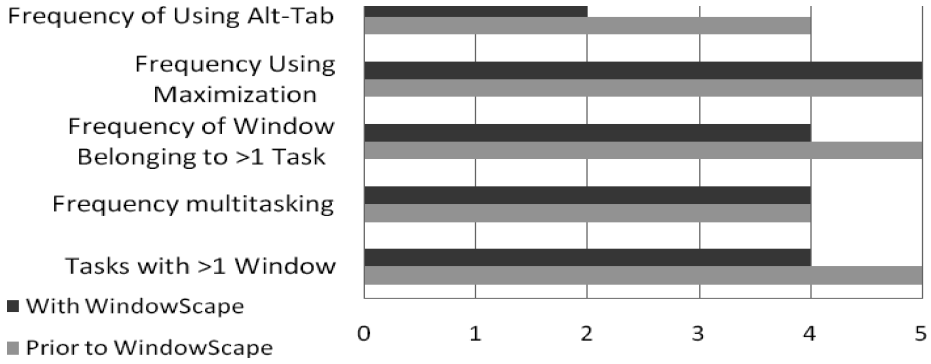

Fig. 17. Median responses to Likert-scale questions (1 to 5, 5 is most frequent) about user behavior.

overlapping groups, that is, where one window is a member of multiple groups. This may be because the need for such support was uncertain, and a viable representational approach that would be compatible with overlapping groups may have been elusive. Our results shed light on this problem, though, first by indicating that users do appear to need overlapping groups, as participants reported frequently using the same window in multiple projects or activities (Figure 17). Second, the results suggest the viability of our grouping model for addressing this problem. Our use of photograph-like snapshots as group representations naturally enables groups to overlap, but we wondered whether users would understand them. Our results suggest that 11 of 13 did, with many readily creating groupings and describing the groupings as very useful. This raises the prospect that, if embedded in the proper metaphor, users can understand many-to-one relationships between proxies and the windows they represent. Other grouping systems could thus use similar techniques to support commensurately flexible definitions of groups. More work will be needed however, to evaluate how less expert users react to this approach to grouping, and to understand why even some very experienced users had difficulty with it.

More broadly though, our discussions with participants strongly suggest that WindowScape, or something like it, is desirable in at least some circumstances. Of the 12 interviewed participants, 8 felt that more than the standard desktop was needed, one of whom said that this need has been filled by VDMs. Three others felt that novel window management tools were not necessary per se, but would be helpful in certain cases. All of these participants were very skilled computer users who had likely been exposed to a variety of supplementary window management tools, but yet very few used them, and only one made any mention of such tools as actually satisfying that need for better window management. One participant even pointed out that she was not sure why she did not use a VDM. Naturally, this raises the question of why these tools have not enjoyed broader adoption.

As a preliminary step toward understanding this behavior, we observe that WindowScape users in our study tended to group together fairly small numbers of windows, an average of about 1.9. With such small groups, the division of one's workspace into complete, separate desktops, as with VDMs, may be excessive. Indeed, many participants said they liked WindowScape because it reduced the feeling of clutter and helped them maintain awareness of their windows. Thus, rather than completely isolating windows from one another, it may be more expedient to focus on lighterweight approaches to window organization that maintain peripheral cues for other windows, in the vein of projects like GroupBar and Scalable Fabric, as well as WindowScape [Smith et al. 2003; Robertson et al. 2004].

But it is important to include in any discussion of user behavior and needs that users exhibit considerable diversity in these areas. Our study demonstrated this in several ways, the most striking of which was in participant reactions to WindowScape's timeline and bookmark bars. As discussed earlier, reactions to the timeline ranged widely, with some participants finding it useful and others seeing it as virtually useless. Similarly, users reported a variety of approaches for organizing their windows, both in terms of the spatial position of the miniatures as well as the logical groupings of the windows. One point which we observe here is that the organizational needs of users vary widely, and thus the organizational tools given to users must be flexible and allow for a variety of approaches to organization, including organization based on criteria other than task or activity. This supports the results of both Oleksik et al. [2009], who also found a remarkable variety of criteria with which users sought to organize their work in the context of a window and file tagging mechanism, as well as research showing the wide variety of organizational strategies applied to traditional, full-sized windows [Hutchings and Stasko 2004].

But although participant reactions to the timeline varied among those who understood its purpose, several users did not understand how it could be of use at all. We believe this highlights one of the challenges inherent in software that affords highly unfamiliar functionality: users need an understanding of the scenarios where it can be of value, a need which one user made explicitly during our interview. This suggests that as experimenters, our role may sometimes be more than just that of observers. To some extent, we need to take on the role of teachers, educating our participants about any nascent best practices we have observed or developed around our intervention. Doing this does entail the risk that we will modify our subjects' behaviors by informing them of scenarios, rather than letting them emerge. But from our experience, users in voluntary study contexts who do not see how to make an intervention useful right away may simply abandon it. Ultimately, it appears unclear how and when best to present use scenarios, with a variety of trade-offs for any given choice. A potentially valuable area for future research then, may be to further explore the issue of scenarios in deployments and what impact they have on behavior.

Apart from window grouping functionality, we were interested in the strongly positive reaction to the general window manipulation operations in WindowScape. Indeed, the interviews suggested participants were actually more receptive to the functions for miniaturization and retrieving occluded windows than those for the window grouping. The usage logs reflected this, showing the general window management facilities being used far more frequently. This has several implications, but perhaps the most important is that the commercial state-of-the-art appears to be inadequate. Users' needs for window manipulation appear to extend beyond just grouping; indeed, we may want to reconsider how important grouping may ultimately be. For example, it appears that other means for expediting window retrieval, providing peripheral awareness, and supporting a sense of organization do indeed represent important research directions as well. Encouragingly, the research community is exploring these issues (e.g., Beaudouin-Lafon [2001] and Bernstein et al. [2008]) and we believe our results suggest that these issues are still very present and need continued attention.

Our study does also, however, provide evidence that certain window management techniques might help to address these issues. Window miniaturization, for example, generally received a very positive response from users. This is particularly significant because, although there have been many studies of window thumbnails (e.g., Robertson et al. [2004], Bardram et al. [2006], and Kumar et al. [2007]), there have been very few that evaluate their utility in the setting of a naturalistic deployment (e.g., a small study in Robertson et al. [2004]). Thus, we were interested to observe that most users were very pleased with miniature representations of their windows, typically not reporting

difficulty distinguishing or managing them. Users generally felt the system allowed them to find windows more easily, and some specifically noted the benefit of a spatial organization; this confirms the assumptions of prior work in related projects [Robertson et al. 1998; Robertson et al. 2004]. We believe this indicates that some form of window miniaturization can be a viable part of future window management functionality. And while our study shows thumbnail views occasionally do suffer distinguishability problems, research exploring alternate thumbnail representations may help address these problems in certain contexts [Woodruff et al. 2002; Matthews et al. 2006].

A more general observation on our study was the considerable sensitivity to which deployment evaluations are prone. Unfortunately, participant experiences, and thus reactions, to WindowScape tended to be strongly degraded by even small problems with the user experience. For example, one user found that WindowScape made the buttons on Windows' taskbar blink after he miniaturized a window. This problem, which other users did not experience or did not notice, bothered this individual enough to dissuade him from using WindowScape. We found other users were strongly influenced, if not quite as extremely, by other shortcomings that are relatively minor from a theoretical perspective, but very substantial from a practical perspective. As such, this sensitivity of deployments suggests that care must be taken in evaluating their results, disentangling the verdict on the principles of a system from a verdict on the details of its implementation.

8. CONCLUSION

WindowScape is a task-oriented window manager that employs photograph-like snapshots, enabling users to implicitly group their windows according to task. Since introducing an early prototype of this system in Tashman [2006], we have conducted a series of formative studies followed by a deployment study in which we evaluated a significantly revised version of WindowScape by giving it to 13 people to use in their day-to-day work.

In this article, we have first reported on the results of these formative studies, and how the design of WindowScape evolved in response. We concluded with a discussion of our deployment study, including user reactions to particular aspects of WindowScape, as well as general results relevant to task and window management.

Through our studies, we found that the use of a photograph metaphor to represent groups was well understood by participants, and that many of them found the window grouping functions useful. However, users also employed a variety of task management behaviors, with different users leaning toward different organizational schemes.

Most users had positive reactions to the scaling window manager, often citing it as their favorite feature of the tool. Many felt it helped them to be better organized and find windows more easily, and few reported difficulties distinguishing between miniatures.

Generally, organizing and finding overlapping windows appears to remain a challenge for users, suggesting that research in window management must continue to expand. Our study suggests that, while grouping can be a valuable aspect of window management, users are interested in a variety of facilities for retrieving and visualizing their windows, which may themselves need to change significantly between users. Continued research will be required to balance this need for variety against the simplicity and consistency users similarly require.

REFERENCES

BARDRAM, J., BUNDE-PEDERSEN, J., AND SOEGAARD, M. 2006. Support for activity-based computing in a personal computing operating system. In *Proceedings of the Conference on Computer-Human Interaction (CHI'06)*. ACM Press, New York.

Beaudouin-Lafon, M. 2001. Novel interaction techniques for overlapping windows. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST'01)*. ACM Press, New York.

Bellotti, V., Ducheneaut, N., Howard, M., and Smith, I. 2003. Taking email to task: The design and evaluation of a task management centered email tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, New York.

Bernstein, M. S., Shrager, J., and Winograd, T. 2008. Taskposé: Exploring fluid boundaries in an associative window visualization. In *Proceedings of the Annual ACM Symposium on User Interface Software and Technology (UIST'08)*.

Bly, S. A. and Rosenberg, J. K. 1986. A comparison of tiled and overlapping windows. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, New York.

Chapuis, O. and Roussel, N. 2005. Metisse is not a 3D desktop! In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST'05)*. ACM Press, New York.

Czerwinski, M., Horvitz, E., and Wilhite, S. 2004. A diary study of task switching and interruptions. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'04)*. ACM Press, New York.

Dragunov, A. N., Dietterich, T. G., Johnsrude, K., McLaughlin, M., Li, L., and Herlocker, J. L. 2005. TaskTracer: A desktop environment to support multi-tasking knowledge workers. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI'05)*. ACM Press, New York.

Feiner, S., Nagy, S., and van Dam, A. 1982. An experimental system for creating and presenting interactive graphical documents. *ACM Trans. Graph. 1*, 1, 18.

Goldberg, A. 1984. *Smalltalk-80: The Interactive Programming Language.* Addison-Wesley, Reading, MA.

Gwizdka, J. 2002. Reinventing the inbox: Supporting the management of pending tasks in email. In *CHI'02 Extended Abstracts in Human Factors in Computing Systems*. ACM Press, New York.

Henderson, A. and Card, S. 1986. Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Trans. Graph. 5*, 3, 211–243.

Hutchings, D. R. and Stasko, J. 2004. Revisiting display space management: Understanding current practice to inform next-generation design. In *Proceedings of the Graphics Interface Conference*. Human-Computer Communications Society.

Kandogan, E. and Shneiderman, B. 1997. Elastic windows: Evaluation of multi-window operations. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'97)*. ACM Press, New York.

Kaptelinin, V. 2003. UMEA: Translating interaction histories into project contexts. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'03)*. ACM Press, New York.

Kumar, M., Paepcke, A., and Winograd, T. 2007. EyeExposé: Switching applications with your eyes. Tech. rep. 7, HCI Group, Stanford University, Stanford, California.

MacIntyre, B., Mynatt, E. D., Voida, S., Marius, K., Tullio, J., and Corso, G. M. 2001. Support for multitasking and background awareness using interactive peripheral displays. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'01)*. ACM Press, New York.

Mark, G., Gonzalez, V. M., and Harris, J. 2005. No task left behind?: Examining the nature of fragmented work. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'05)*. ACM Press, New York.

Matthews, T., Czerwinski, M., Robertson, G., and Tan, D. 2006. Clipping lists and change borders: Improving multitasking efficiency with peripheral information design. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'06)*. ACM Press, New York.

Myers, B., Hudson, S. E., and Pausch, R. 2000. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact. 7*, 1, 3–28.

Myers, B. A. 1988. A taxonomy of window manager user interfaces. *IEEE Comput. Graph. Appl. 8*, 5, 65–84.

Oleksik, G., Wilson, M. L., Tashman, C., Mendes Rodrigues, E., Kazai, G., Smyth, G., Milic-Frayling, N., and Jones, R. 2009. Lightweight tagging expands information and activity management practices. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'09)*. ACM Press, New York.

Rattenbury, T. and Canny, J. 2007. CAAD: An automatic task support system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'07)*. ACM Press, New York.

Rekimoto, J. 1999. Time-Machine computing: A time-centric approach for the information environment. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'99)*. ACM Press, New York.

Ringel, M. 2003. When one isn't enough: An analysis of virtual desktop usage strategies and their implications for design. In *CHI'03 Extended Abstracts on Human Factors in Computing Systems*. ACM Press, New York.

ROBERTSON, G., CZERWINSKI, M., LARSON, K., ROBBINS, D. C., THIEL, D., AND VAN DANTZICH, M. 1998. Data mountain: Using spatial memory for document management. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'98)*. ACM Press, New York.

ROBERTSON, G., HORVITZ, E., CZERWINSKI, M., BAUDISCH, P., HUTCHINGS, D., MEYERS, B., ROBBINS, D., AND SMITH, G. 2004. Scalable fabric: Flexible task management. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI'04)*. ACM Press, New York.

ROBERTSON, G. G., VAN DANTZICH , M., ROBBINS, D. C., CZERWINSKI , M., HINCKLEY, K., RISDEN, K., THIEL, D., AND GOROKHOVSKY, D. 2000. The task gallery: A 3D window manager. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'00)*. ACM Press, New York.

SMITH, G., BAUDISCH, P., ROBERTSON, G., CZERWINSKI, M., MEYERS, B., ROBBINS, D., HORVITZ, E., AND ANDREWS, D. 2003. GroupBar: The TaskBar evolved. In *Proceedings of the OZCHI'03 Conference*.

TAK, S. AND COCKBURN, A. 2010. Improved window switching interfaces. In *Proceedings of the 28th International Conference Extended Abstracts on Human Factors in Computing Systems*. ACM Press, New York.

TAK, S., COCKBURN, A., HUMM, K., AHLSTROEM, D., GUTWIN, C., AND SCARR, J. 2009. Improving window switching interfaces. In Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part II. Springer. In *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction* Part II. Springer.

TASHMAN, C. 2006. WindowScape: A task oriented window manager. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'06)*. ACM Press, New York.

WOODRUFF, A., ROSENHOLTZ, R., MORRISON, J. B., FAULRING, A., AND PIROLLI, P. 2002. A comparison of the use of text summaries, plain thumbnails, and enhanced thumbnails for Web search tasks. *J. Amer. Soc. Inf. Sci. Technol. 53*, 20, 172–185.

XU, Q. AND CASIEZ, G. 2010. Push-and-Pull switching: Window switching based on window overlapping. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems*. ACM Press, New York.

Received August 2009; revised February 2011; accepted October 2011