Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings

Elin Rønby Pedersen, Kim McCall, Thomas P. Moran, Frank G. Halasz*
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304, U.S.A.
Email: elin@dat.ruc.dk, {mccall,moran,halasz}@parc.xerox.com

ABSTRACT

This paper describes Tivoli, an electronic whiteboard application designed to support informal workgroup meetings and targeted to run on the Xerox Liveboard, a large screen, pen-based interactive display. Tivoli strives to provide its users with the simplicity, facile use, and easily understood functionality of conventional whiteboards, while at the same time taking advantage of the computational power of the Liveboard to support and augment its users' informal meeting practices. The paper presents the motivations for the design of Tivoli and briefly describes the current version in operation. It then reflects on several issues encountered in designing Tivoli, including the need to reconsider the basic assumptions behind the standard desktop GUI, the use of strokes as the fundamental object in the system, the generalized wipe interface technique, and the use of meta-strokes as gestural commands.

INTRODUCTION

Most of the early attempts to build computer support for meeting rooms, such as the Colab project [11], included whiteboard-sized displays to provide a shared focus of attention for the meeting participants. However, these displays were controlled at a distance by keyboard and mouse (just like a workstation). Meeting participants did not interact directly with the display surfaces as they would with ordinary whiteboards. Empirical studies of collaborative work at shared drawing surfaces [2,12] show that the physical actions around the surface itself are as important as the actual marks made on the surface to support the collaboration. Thus, the *Liveboard* system [7] was created at PARC to provide a "directly interactive, stylus-based, large-area display."[1] The Tivoli project was started to develop software for the Liveboard technology addressing two goals: (1) to discover and explore the user interface techniques appropriate to this new kind of technology and (2) to discover and implement the functionality needed to actually support small working meetings at the Liveboard.

Nature of the User Interface

The basic direction was influenced by our early Liveboard experiences. After a dozen Liveboards had been deployed throughout PARC in meeting rooms and common areas, the most widely-used application was *Whiteboard* [7], a simple multipage pen-based image editor. It simulated a whiteboard by allowing freehand drawing and erasing. This system provided for the kind of unselfconscious, freeform scribbling that is prevalent in small group interactions, and it confronted us with the issues of the appropriate interaction techniques for and the basic nature of this genre of user interface.

The basic function of the board is to support interaction between people (and in this we agree with [4] that it is a "conversation board"). As such, the most important criteria for the user interaction with the board are to be *unselfconscious* — so as not to draw the attention of the participants from their interaction with each other — and to be *fluid* — to allow unhindered expression of ideas. While it is true that the user's "text-graphic performance" [9] involves the creation of various kinds of text-graphic objects, it does not follow that users should be required to declare these as they would be by a structured drawing program like MacDraw. In this sense we disagree with [4]. Further, we deliberately do not provide for handwriting recognition, for it would be disruptive in this context.

The Liveboard, as one component of a vision of ubiquitous computing [13], "blends into the woodwork" and appears to be just a familiar whiteboard. To follow through on this, the board should also behave as a simple whiteboard, at least initially. This not only allows first-time users immediate use of the board, but also allows users to build from their current work practices involving whiteboards.

Pen-based computing is still in its infancy. Operating system support, such as PenPoint [6] and Microsoft Windows for Pen

1. Other large-surface displays are also becoming available, such as the Smart Technologies 2000 and the WACOM Meeting Staff.

*Elin R. Pedersen's current address is: Department of Computer Science, Roskilde University, DK-4000 Roskilde, Denmark.

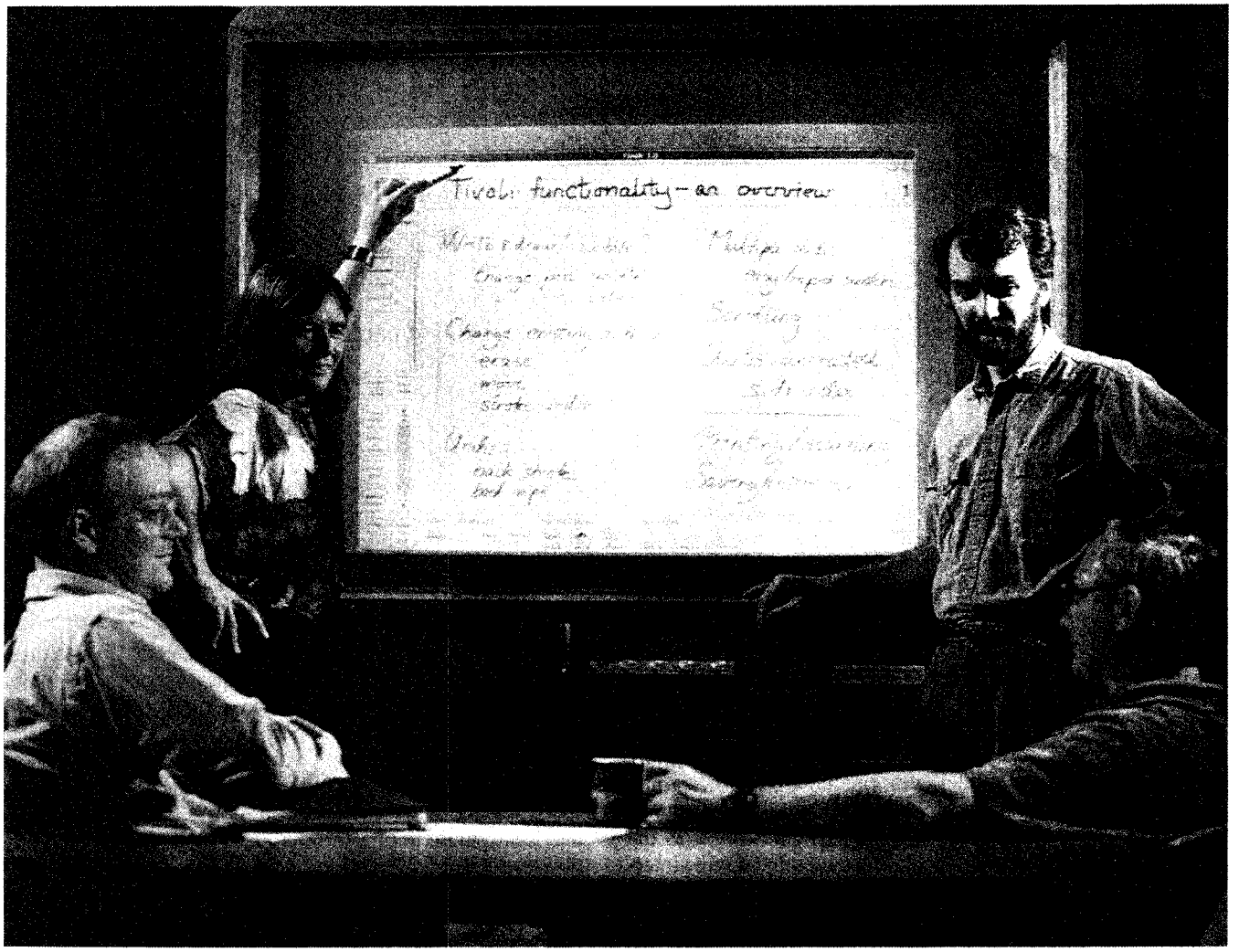

Figure 1: Photograph of Tivoli running on a Xerox Liveboard being used in an informal meeting.

Computing, is just now being provided. We want to discover appropriate ways to utilize the pen as an input device. The motions that are natural for a pen are quite different from those for a mouse. Because we are operating on a large surface, we want to keep as much facility "at hand" as possible. Gesturing (i.e. creating marks to be interpreted as commands) is an obvious extension beyond drawing and erasing. Other pen-based systems are oriented to using marks as annotations and gestures as commands over structured material, such as form-filling applications. Our situation is somewhat different in that all our material consists of informal handmade marks, and we want to preserve this look and feel. This raises new user interface issues that we are beginning to address.

Functionality for Small Working Meetings

Tivoli is targeted at supporting small working meetings, because this kind of meeting activity is prevalent at PARC and is well suited to the Liveboard. By "working" meetings (as opposed to, say, presentations) we mean meetings where the participants work closely together on a problem or an idea. For these a scribbling surface is useful for generating, communicating, and clarifying ideas and for keeping track of information. Such a working group usually needs to be small: up to about 8 people (also, the physical size of the Liveboard image, 46" x 32", precludes more people being able to gather around it effectively). Because meetings are small and informal, there is no official facilitator or scribe; all participants can have access to the board.

Our goal with Tivoli was to extend the functionality of a simple whiteboard program while preserving its basic features, spirit, and style. The early ideas for new functionality were numerous and diverse — much more than we could possibly implement. As a design strategy to sort these out, we developed a series of 14 scenarios of various kinds of meetings — varied both in content (design, review, brainstorming, administrative meetings) and in style (small to medium, informal to structured, co-located and distributed meetings). Each scenario was a 2-9 page narrative description of a meeting and its use of available wall displays, including whiteboards, flipcharts, copyboards, and Liveboards. Half the scenarios were *documentaries* of actual meetings, and half were *envisionments* of possible meetings. Writing these scenarios served to

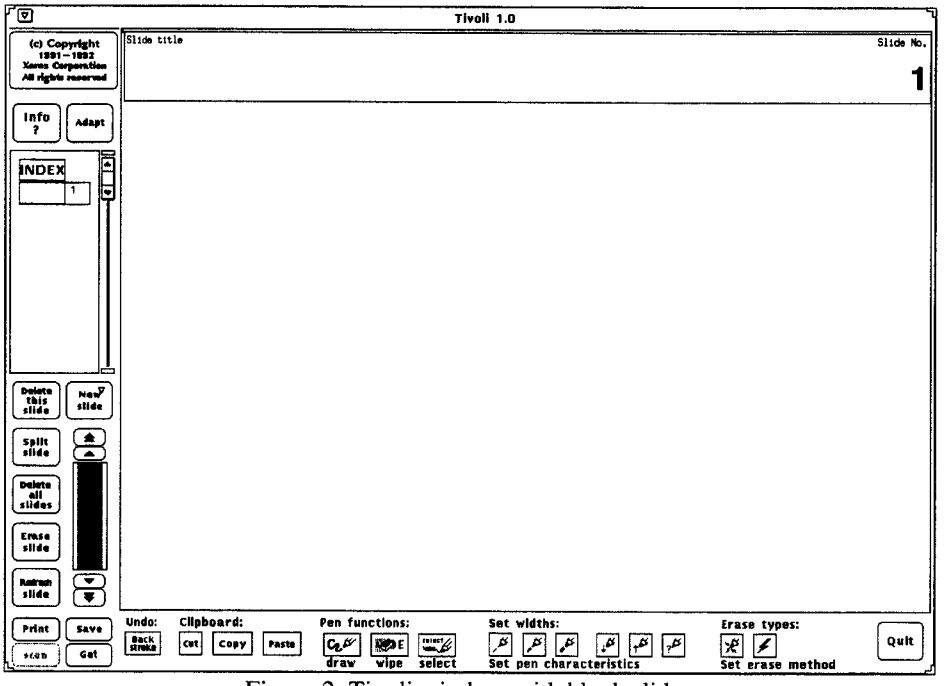

Figure 2: Tivoli window with blank slide.

integrate and concretize inputs from a variety of people, including both designers and potential users.

Analysis of these scenarios resulted in the identification of a coherent *core* of functionality needed across all meeting types and the classification of the remaining functionality into different research *thrusts*. The core functionality included simple pen and gesture scribbling and editing, multiple pages, saving and retrieving, printing, and importing images. The thrusts involved remote collaboration (shared drawing), meeting management tools, and integration with other "ubiquitous" devices. Tivoli was designed to implement the core functionality on an architecture that anticipated the explorations into the extended functionality. In this paper, we describe and discuss Tivoli 1.0, the first release that implements the core functionality.

A SCENARIO OF TIVOLI USE

The following scenario illustrates the main features of Tivoli 1.0 and motivates many of the design issues discussed in the next section. Elin, Kim, Tom, and Frank are starting a design meeting in which they intend to discuss the Tivoli object hierarchy and the possibility of implementing the "decorations" that appear on the Tivoli drawing surface as objects within that hierarchy. Elin goes to the Liveboard, which is displaying the Tivoli window shown in Figure 2. The window at this point consists of a blank *slide* for drawing in and control panels along the bottom and left edges.

In a long box at the top of the slide Elin writes "Objects in the Tivoli World." She does this as naturally and directly as if she were writing with a marker on a whiteboard. Then she lists the main classes of user-generated objects, "Stroke," "Character," and "Image." Kim points out that "Images" are now called "Pictures," so Elin taps the **wipe** button at the bottom of the window. The pen's cursor turns into an eraser. As she sweeps the pen over the word "Images," each stroke she'd drawn is erased as a unit as soon as she touches it. She taps the **draw** button, sees her usual drawing cursor again, and writes "Picture."

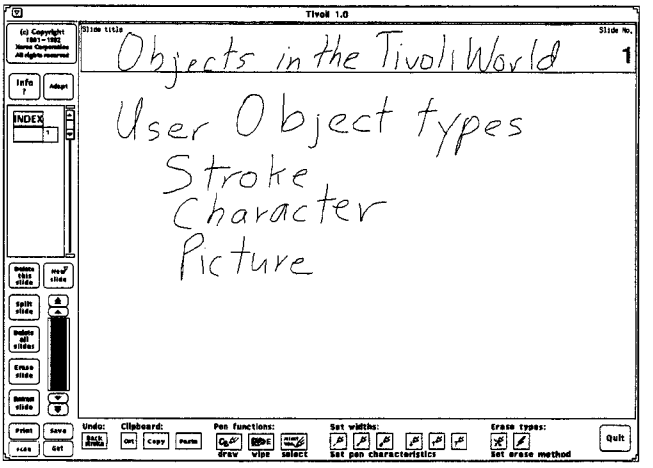

Figure 3

Elin wants a heading for these terms, but needs to create room above them to write it. She taps on the **select** button and draws a circle around the three terms, which then become selected. Then, she draws a line to show how far she wants to have the selected objects moved, which they do when she lifts up the pen. Above them she writes "User Object types." The result is shown in Figure 3.

Tom complains that Elin has been writing with too fine a pen. So she circles her list again and taps on the button for a thicker pen width. All the selected strokes get thicker as can be seen in Figure 4. But she failed to completely include the first 'U', which she deals with separately. But she has still left out the title at the top. She again taps the **wipe** button. But rather than just sweeping across the title (which would erase it), she

first "dips" the pen into the button representing the preferred pen width. Now as she wipes it across the title strokes, they are all repainted at the new width. A tap of the **draw** button gives her back her regular pen.

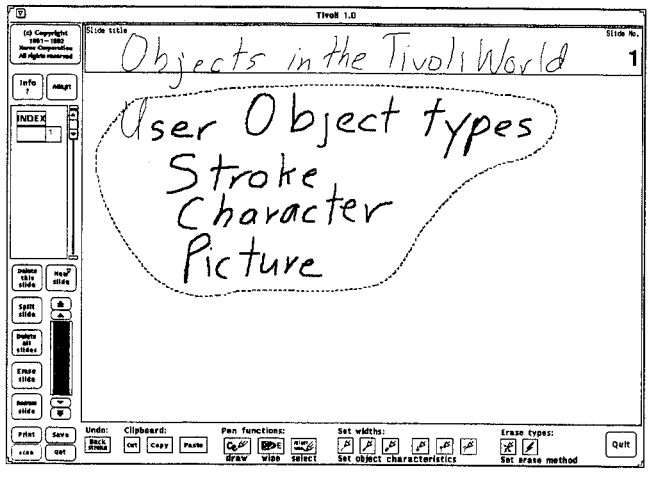

Figure 4

Elin starts a list of system-generated objects or slide "decorations." The list reaches the bottom of the slide, but Frank mentions a couple of items that were left out and really belong near the top of the list. Elin draws a horizontal line across the slide (what we call a "tear" gesture) where she wants more room and quickly taps the pen twice. Everything below the line gets selected, and she moves it down. Then she writes the desired items in the space just opened up. Some of the list is no longer visible, so she taps a small arrow button to scroll the slide. A scroll indicator near the arrow reflects where the current viewport is on the entire slide. The scrolled slide is show in Figure 5.

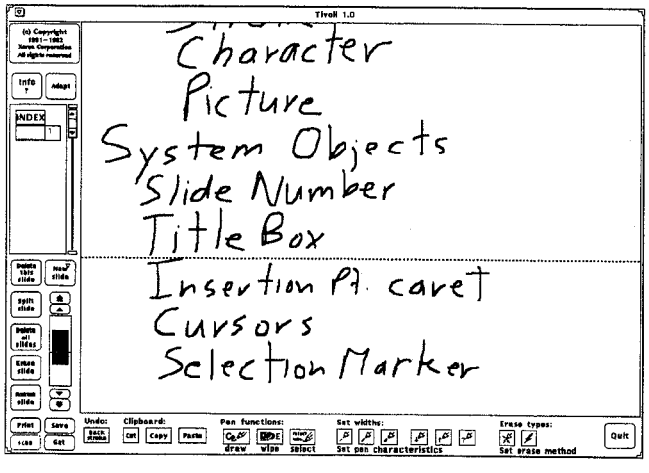

Figure 5

Now it's time to list issues concerning the objects, and Elin decides to start listing them on a separate slide; so she taps the **New Slide** button and gets a blank slide. At the top she writes "Issues" and then lists several issues as people mention them. A *slide list* to the left of the slide area contains a numbered list of all of the slides she has created. She taps on the numbers to switch between the two slides.

The group gets embroiled in a debate on the virtues of special-case representations and display routines. Elin is taking notes on the arguments, but soon decides this discussion belongs on a separate slide; so she selects it all and cuts it with a pigtail gesture. She then creates a new slide and taps the **Paste** button. The strokes that had been cut show up on the new slide. They remain selected, as seen in Figure 6, inviting her to move them, which she does.

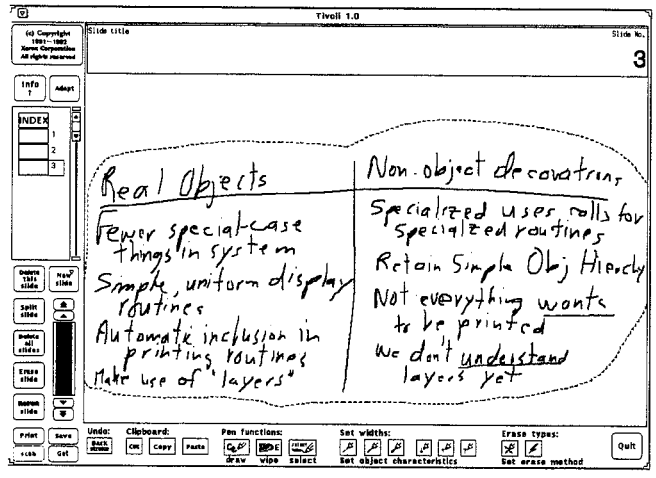

Figure 6

As the discussion starts to get very technical, Frank begins to worry about its implications for the display and object management aspects of the implementation. Kim remembers that he once prepared a slide for a formal talk illustrating the implementation modules. He goes to the Liveboard, taps the **Get** button, and selects from a dialog window the file containing his slides for that talk. Those slides are all added into the current folder of slides, which now number eight.

Unsure of which of the five new slides was the desired one, Kim taps the **INDEX** entry at the top of the slide list. This produces a system-created slide which consists of the title areas from the other slides, as shown in Figure 7. He sees that the desired Architecture slide is number seven. He thinks about just going to that slide, but decides instead to clean things up by deleting the unwanted slides. He uses the slide index to go to each slide in turn and delete it by tapping the **Delete this slide** button.

Then he goes to the Architecture slide and starts annotating it with circles and arrows. The resulting slide is shown in Figure 8. Occasionally he erases some of these, toggling back and forth between drawing and erasing by a rapid tap-tap of the pen. Because the illustration is in the *background* it is indelible. Only the annotations get erased. At one point he erases too much, deleting one arrow too many. He taps on the **Back stroke** button until he has recovered the lost arrow.

Elin wants to propose a new twist, but Kim is monopolizing the group's attention, so Elin draws a picture on a piece of paper before the idea escapes her. When the opportunity presents itself she sticks the paper into the Liveboard's attached scanner, taps the **Scan** button, and waits a minute. Then she

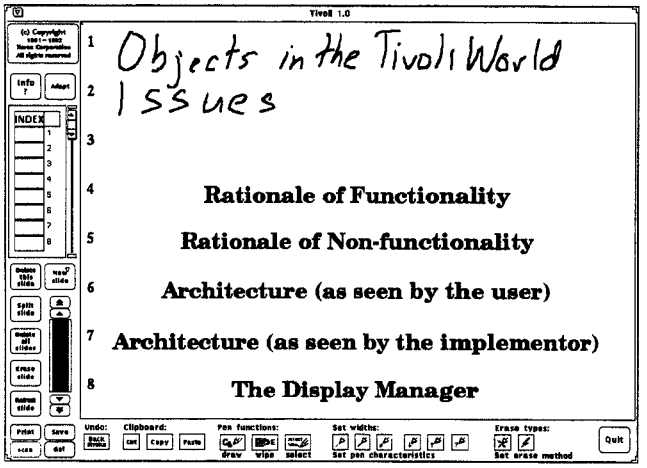

Figure 7

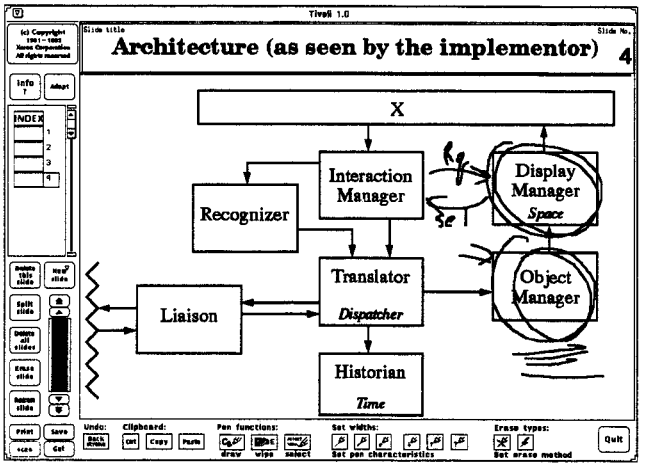

Figure 8

presses a button that imports the scanned image as the background of the current slide. Then she explains it and further annotates it.

Before anything is decided, the meeting's time has expired. Frank wants a hardcopy of the slides, so he taps the **Print** button, selects the **4 slides per page** option so that they'll all show up on one sheet, and sends it off to the printer. Tom is nominated to see whether he can condense the discussion to anything actionable. Frank finally taps the **Save** button. A dialog window comes up that suggests a default file name under which to save it. But Frank thinks up a better one, and *for the first time in the entire meeting* he uses the Liveboard's attached keyboard to specify the desired file name.

A few hours later, Tom runs Tivoli on his office workstation and reads in the folder in order to review the meeting. Imagining that this folder will form the basis for the next meeting, he decides to create another slide summarizing outstanding issues. But since his workstation has no stylus for drawing, he decides to do most of his writing with the keyboard.

The group's next meeting starts by reading in and reviewing Tom's revised version of the folder.

DESIGN FEATURES AND ISSUES

We now discuss some of the more interesting design features of Tivoli and the user interface design issues that they raise.

Assumptions Reconsidered

User interface design has converged onto a broad set of principles defining the generic graphical user interface (GUI). However, the GUI is tied to a set of assumptions about the technology (such as the mouse and the keyboard) and the usage situation (a single user at a workstation on a desk). Both our technology and our usage situation are different for Tivoli, and we found ourselves reconsidering many tacit assumptions. We give three examples:

Pen vs. mouse

A mouse is designed for pointing, whereas a pen is also for writing and drawing. Thus gestural techniques are much more natural for a pen than a mouse. A mouse is indirect in the sense that computation mediates the relationship of the physical movement of the mouse to the movement of the cursor on the display. On the other hand, the Liveboard pen is pointed directly at the surface of the display. There does not necessarily need to be a cursor; it should be where the pen is pointing (within the limits imposed by calibration error). This directness provides for a more direct interaction with objects on the display. However, the system needs to respect this physical directness. For example, we use an OpenLook scroll bar to implement the slide list. When the user points to a scroll arrow, the scroll arrows ride up and down the scrollbar like an elevator. With the mouse this is fine, because the cursor is adjusted to ride the elevator. But with the pen the elevator slides out from underneath the stationary pen, throwing the user off the scroll arrow.

Very Large Interactive Surface

Even with a large workstation display, the user can have a reasonable overview of the display from the normal viewing distance. A user standing at a Liveboard, however, is too close to have such an overview. Thus, we have had to adjust the size and placement of pop-up messages to make sure the user will always notice them. Also, tools such as buttons and menus can be physically out of the user's reach on the Liveboard. (This becomes apparent when running off-the-shelf workstation software on the Liveboard.) This implies some obvious design constraints, such as not requiring users to reach the top of the display to press a button. More generally, it places greater emphasis on keeping user control at the pen, especially for frequent actions and for actions that occur within the flow of scribbling. This is another motivation for utilizing gestural commands, which are issued from the pen *at* the place of application. The problem with gestures is that novice users may not remember them, and so we provide redundant screen buttons where possible. For example, the user can toggle between drawing and erasing either by a double tap of the pen or by reaching down and tapping the **draw** and **erase** buttons at the bottom of the display.

Multiple Users and Pens

A Liveboard is large enough to have more than one person working at it at a time — a natural working situation [11, 12]. The Liveboard is designed to have three pens to accommo-

date multiple users, and Tivoli was designed to operate with multiple pens. One consequence is that tools on the display are shared. For example, we cannot highlight an icon to show the current pen line thickness, since it can be different for different pens. This is yet another argument for gestural actions, which are localized to specific pens and do not have to rely on shared on-screen tools. In general, in our implementation we have had to distinguish *pen state* from *system state* (see also [3]). This distinction is not always clear-cut. For example, we consider a selection of objects to be part of a pen's state, and thus a second user cannot operate on the first user's selection (without taking the first pen). Another example is that we mark all actions with the ID of the pen that produced them, and we restrict the undoing of actions to be by the same pen (i.e, pen_2 cannot undo the relocation of strokes performed by pen_1, although pen_2 can erase those strokes).

Atomic Objects

Tivoli has an informal look and feel that is much like that of paint programs. But a major feature of the Tivoli design that distinguishes it from paint programs is that drawings are represented as *stroke objects*, not as pixel map images. A stroke is simply a series of line segments describing the path of the pen from the time it touches down on the display surface until it is lifted. A stroke is created over time, and the system cannot wait until it is completed before displaying it (as it does with keystrokes). Therefore, we provide an "inner loop" of feedback that instantly inks the display surface as the pen is moved over it. Only after the pen is lifted does the system create a stroke object.

The stroke-level granularity of this representation manifests itself when operating on the drawing. For example, erasing is done by wiping over the surface with the pen in erase mode. As the eraser goes over each stroke, that whole stroke disappears (i.e., the stroke object is deleted from the representation). This has proved to give a very nice "power boost" when erasing small to medium size strokes, such as handwriting (sweeping across a word gets rid of it without having to fuss with every little pixel). But this sometimes causes surprises when a long stroke (e.g., a large box or a long arrow) disappears when only a small portion of it has been touched by the eraser. The general issue is that objects are not homogeneous, and operations need to be designed to have a desirable effect. This issue needs much further exploration.

Another advantage of strokes as atomic units is that they provide objects for holding properties (e.g., color and thickness). Stroke properties can be changed, e.g., the color of a whole stroke can be changed, no matter how entangled it is with other strokes. And because they are geometrically defined, strokes are transformable.

Generalized Wiping

Tivoli expands the notion of object-based erasing to generalized *wiping*. Wiping is a user's back-and-forth motion across the display surface with a pen, which in wipe mode is called a *wiper*. The wiping motion *dynamically* selects objects as the wiper passes over them. A wiper has an associated operator, which it applies to objects as it selects them. Erasing is accomplished when the wipe operator is DELETE. When the user enters wipe mode, the default operator is DELETE; and thus a novice user can simply think of this as an eraser. But the wipe operator can be changed just by tapping any on-screen button with the wiper. For example, tapping the red button changes the wipe operator to MAKE-RED, after which wiping makes all selected objects red. This has proved to be a natural and powerful technique for dynamically selecting and applying operations to objects.

Gestures (Meta-Strokes)

We have noted several motivations for the use of gestural techniques with the pen. By the term "gesture" we mean a *meta-stroke*, i.e., a stroke that is not just taken to be an element of the drawing, but is to be *interpreted as a command.* A gesture is drawn as a regular stroke and then interpreted. This has the beneficial effect in a group situation that the non-drawing participants can see the gesture before the action takes place and thus not be surprised by unexpected behavior. The most common gesture is the selection gesture, which is drawn as a closed loop of arbitrary shape around a set of objects, defining them as the *selection*. Subsequent gestures then operate on the selection: a pigtail gesture deletes the selected objects; a stroke that starts inside the selection loop moves the selection to the other end of the stroke (the encircled objects remain selected so that they can be operated on again).

One of the main design issues is how to indicate that a stroke is a gesture. There are several classes of techniques, involving devices (e.g., a gesture pen vs. a drawing pen), modes (e.g., gesture mode vs. drawing mode), "tense modes" [5] (e.g., holding down button on the pen to indicate a gesture), and so on. We attempted to avoid modes in the first version of Tivoli, and so we explored a different technique: the *postfix indicator*, which converts the preceding stroke into a gesture. For example, to select and delete a word, first draw a loop around the word, then give the indicator (which repaints the loop in the distinguished gesture color), then draw a pigtail.[2] The postfix technique allows gestures to be embedded in the drawing process. For example, a loop can be drawn (which at this point is just a regular stroke) and, if it isn't correct, erased and redrawn before indicating that it is to be interpreted as a gesture.

We explored a *double tap* of the pen as the postfix indicator (by analogy to a double-click on a mouse button). There is an inherent ambiguity with a double tap during drawing: how to distinguish it from two intentionally small drawing strokes, such as a colon or a quote mark. Thus a user writing "To:" might find that the system attempts to interpret the "o" as a gesture (unsuccessfully because the "o" is an empty loop) and all that is left is a "T".[3]

Faced with this difficulty in distinguishing double taps, especially for users with a "choppy" style of handwriting, we have moved to using a pen button as the gesture indicator in the more recent versions of Tivoli.

2. The pigtail does not require an indicator since it is indicated implicitly by context: When a selection is made, the system expects the user to next operate on it; and thus any stroke starting within the selection loop is taken to be a gesture.

STATUS AND FUTURE DIRECTIONS

Tivoli was first released in December 1991 and since then has been installed as the primary application on the Liveboards at PARC. It runs on Sun Sparcstations and utilizes the X Window System. It is implemented in C++ and currently uses the XView toolkit to implement control panels and buttons.

Tivoli 1.0 is only the first step in our attempt to build a collaborative meeting tool for the Liveboards. Among the many improvements and extensions we hope to eventually explore, we are currently focusing our effort on three issues: (1) a multi-site, shared version of Tivoli; (2) making the interface more accessible, or perspicuous, for casual users; and (3) exploring how Tivoli can be used to support specific meeting practices.

Multi-Site Tivoli

Our vision is to make Tivoli a true collaborative tool, supporting both multiple users standing shoulder-to-shoulder around a single Liveboard and multiple users utilizing two or more geographically-separated Liveboards (This draws inspiration from other work on shared drawing systems, such as [2, 8]. The version of Tivoli described in this paper is implemented to support only the single Liveboard case. We have implemented a multi-site version of Tivoli (Tivoli 2.0) that preserves all of the functionality in the current version [10].

Although the current version supports multiple users around a single Liveboard, the majority of our experience to date has been with situations where only a single user operates Tivoli at any given time. It remains an open question whether our current design will provide appropriate support for the multiple user (single Liveboard) case. In particular, the current Tivoli includes little mechanism for coordinating the actions among multiple users (e.g., preventing one user from changing slides while another is drawing), with the idea that such coordination can be accomplished using social rather than technological means. As we gain experience with the multiple user situation, we will be able to assess if this is in fact the case.

Another usage scenario for Tivoli involves a single meeting room containing a single Liveboard communicating with a (pen-based) laptop for each meeting participant. We hope to explore how the principles we are using to design a multi-site Tivoli will apply this co-located, multi-host case.

The Casual User

The design of Tivoli represents a deliberate balance between two often conflicting goals: interface simplicity and functionality. On the one hand, our goal was to design a Liveboard interface that maintains the natural, facile, immediately obvious feel of a conventional whiteboard. In particular, we were interested in maintaining the whiteboard's ease-of-use to casual (or never-before) users. Liveboards were designed to be placed into public meeting spaces. We wanted to make it easy for a person who has never used a Liveboard to walk up and figure out how to accomplish what they could on an ordinary whiteboard (draw in multiple colors, erase) plus a few obvious extensions (change slides, print) with little or no training. Meeting this requirement tended to push the Tivoli user interface towards simplicity, with the control panel displaying only a few straightforward buttons.

On the other hand, the Liveboard is a computational device. Another of our goals in designing Tivoli was to use this computational power to the user's advantage in supporting and augmenting the capture and organization of ideas in meetings. Accomplishing this goal tended to introduce sophisticated features and functionality into Tivoli (e.g., the generalized wipe operations) whose interface was either complex or non-intuitive or both. Moreover, the more such features we added to the program, the more widgets and buttons needed to be added to the interface.

It's clear from the feedback we've received from our user community that our current user interface design has erred in favor of increased functionality over intuitiveness, or perhaps 'interface perspicuity'. The never-before user who walks up to a Liveboard tends to find the interface so complex that she has some difficulty figuring out how to do the basics. We consider this a major problem and are exploring a revised user interface design which improves the walk-up-ability of Tivoli without decreasing the level of functionality.

Support for Specific Meeting Practices

A very explicit goal of the Tivoli project is to develop research prototypes with sufficient functionality and robustness that they can be put into widespread use in real meeting situations. Tivoli 1.0 is the first step in achieving this goal. Tivoli is currently in use on the 10 Liveboards throughout PARC, and we are beginning to accumulate feedback from users engaged in a variety of tasks and meeting types. At minimum, this feedback will guide us in making incremental improvements to Tivoli. More importantly, we are interested in examining in some detail how Tivoli is utilized for specific meeting practices. Our goal is to better understand what functionality could be added to Tivoli in order to support or augment these meeting practices. For example, Tivoli is frequently used to support brainstorming meetings. We are currently exploring how Tivoli could be expanded to support typical brainstorming activities such as list-making, categorization, and voting without altering its fundamental whiteboard-like nature.

In some sense, our search to expand Tivoli to provide support for more specific meeting practices reflects the other side of the tension between a simple, facile, whiteboard-like interface and an application that takes maximum advantage of the computational power behind the Liveboard to support its users' work practices. Clearly, the design challenge for this kind of system is to satisfy both of these goals.

3. Actually, the situation was much worse. The double tap was also loaded with a second function: to indicate a switch between drawing and wiping mode. A double tap would first cause the previous stroke to be interpreted as a gesture; if the gesture recognizer did not recognize it, then it caused a mode jump. Thus, an inadvertent double tap usually threw the user into wipe mode without warning, a dangerous move to say the least! This mode toggling functionality was removed from double tap in subsequent versions.

REFERENCES

1 Sara A. Bly. A Use of Drawing Surfaces in Different Collaborative Settings. *Proceedings of CSCW 88 Conference on Computer Supported Cooperative Work*. Portland, 1988, pp. 250-256.

2 Sara A. Bly and Scott L. Minneman. Commune: A Shared Drawing Surface. In *SIGOIS Bulletin*, Massachusetts, 1990, pp. 184-192.

3 Eric A. Bier, Steve Freeman. MMM: A User Interface Architecture for Shared Editors on a Single Screen. *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST'91, 1991.

4 Tom Brinck, Louis M. Gomez. A Collaborative Medium for the Support of Conversational Props. To appear in *Proceedings of CSCW 92 Conference on Computer Supported Cooperative Work*. Toronto, 1992.

5 William Buxton. Chunking and phrasing and the design of human-computer dialogues. *Proceedings of the IFIP World Computer Congress*, Dublin, Ireland, 1986, pp. 475-480.

6 Robert Carr, Dan Shafer. *The Power of the PenPoint*. Addison-Wesley, 1991.

7 Scott Elrod, Richard Bruce, David Goldberg, Frank Halasz, William Janssen, David Lee, Kim McCall, Elin R. Pedersen, Ken Pier, John Tang and Brent Welch. Liveboard: A large interactive display supporting group meetings, presentations and remote collaboration. *Proceedings of CHI '92 Conference on Human Factors in Computing Systems*, Monterey, CA, 1992.

8 Hiroshi Ishii, Minoru Kobayashi, Jonathan Grudin. Integration of Inter-personal space and shared workspace: ClearBoard design and experiments. *Proceedings of CSCW 92 Conference on Computer-Supported Cooperative Work*, Toronto, 1992, pp. 33-42

9 Fred Lakin, John Wambaugh, Larry Leifer, Dave Cannon, Cecelia Sivard. The Electronic Design Notebook: Performing Medium and Processing Medium. *Visual Computer: International Journal of Computer Graphics*, 214-226, August 1989.

10 Kim McCall, Thomas Moran, Bill van Melle, Elin Pedersen, Frank Halasz. Design principles for sharing in Tivoli, a whiteboard meeting-support tool. *Position paper for the Workshop on Real Time Group Drawing and Writing Tools, CSCW 92 Conference on Computer-Supported Cooperative Work*, Toronto, 1992.

11 Mark Stefik, Greg Foster, Danny Bobrow, Ken Kahn, Stan Lanning, Lucy Suchman. Beyond the Chalkboard: Computer support for collaboration and problem solving in meetings. *Communications of the ACM, 30(1)*, 1987. Also in Irene Greif, editor, *Computer Supported Work: A Book of Readings*, Morgan Kaufmann Publishers, 1988, pp. 335-366.

12 John Tang. *Listing, Drawing, and Gesturing in Design: A Study of the Use of Shared Workspaces by Design Teams*. Ph.D. Thesis. Stanford University, Department of Mechanical Engineering, 1989.

13 Mark Weiser. The Computer for the 21st Century. *Scientific American, Sept. 1991*.