Implicit Structures for Pen-Based Systems Within a Freeform Interaction Paradigm

Thomas P. Moran, Patrick Chiu, William van Melle, Gordon Kurtenbach**

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

{moran,chiu,vanmelle}@parc.xerox.com
gkurtenbach@alias.com

ABSTRACT

This paper presents a scheme for extending an informal, pen-based whiteboard system (Tivoli on the Xerox LiveBoard) to provide a structured editing capability without violating its free expression and ease of use. The scheme supports list, text, table, and outline structures over handwritten scribbles and typed text. The scheme is based on the system temporarily perceiving the "implicit structure" that humans see in the material, which is called a *WYPIWYG* (What You Perceive Is What You Get) capability. The design techniques, principles, trade-offs, and limitations of the scheme are discussed. A notion of "freeform interaction" is proposed to position the system with respect to current user interface techniques.

KEYWORDS: freeform interaction, implicit structure, pen-based systems, scribbling, whiteboard metaphor, informal systems, recognition-based systems, perceptual support, list structures, gestural interfaces, user interface design.

INTRODUCTION

Our goal is to create computational support for the informal collaborative processes of small groups working together in real time. We are concerned especially with "generative" tasks (creating and assessing new ideas and perspectives, discussing them, playing with them, organizing them, negotiating about them, and so on). Human interaction in such situations is informal, freewheeling, rapid, and subtle.

Computational systems are typically ill-suited to such situations, because they force users to create and deal with more-or-less formalized representations. The overhead of using such representations inhibits the very processes they are meant to support [7,12]. One of the big challenges for current HCI design is to create systems for informal interaction.

Pen-based systems that allow scribbling on wall-size displays or notepads can support whiteboard or shared notebook metaphors for interacting with informally scribbled material. The free, easy, and familiar expression permitted by such systems make them a promising class of tools to support informal interaction.

Our base tool is a large, shared, pen-based electronic display device called the *LiveBoard* [4]. We have developed a software system, called Tivoli [9], that simulates whiteboard functionality on the LiveBoard.[1] (There is a commercial version of Tivoli called MeetingBoard.[2]) This paper presents and discusses a new scheme that we have designed and implemented in Tivoli to extend its editing power, while yet remaining simple, natural, and consistent with the informal nature of the tool.

This paper begins by proposing the notion of "freeform interaction" to help pin down what we mean by "informal." Then we describe the extended editing scheme, which is based on the system perceiving the "implicit structure" that humans see in the material. This is followed by a discussion of the design principles, trade-offs, limitations, and comparison to other systems.

FREEFORM INTERACTION

The notion of "informal interaction" is somewhat vague, and so we define a more operational notion. A graphical editing system allows a user to manipulate graphical objects (GOs) that have defined positions in a 2-D space. A *free graphical object* (freeGO) is a GO that has no constraints or structural relations with other GOs; it can be freely operated upon independently of any other GOs in the space. Any kind of GO — such as an ink stroke, a text character, an icon, or a composite GO — can be a freeGO. Typical operations are drawing, erasing, wiping, dragging, and gesturing (for both selecting and operating). A representation consisting solely of freeGOs is a *freeform representation*. The unconstrained

1. Tivoli is written in C++ and runs under Unix and X Windows on Sun-based LiveBoards and on Sun workstations.

2. MeetingBoard is being developed and marketed by Xerox's LiveWorks, Inc. It runs under Microsoft Windows on PC-based commercial LiveBoards and on PC workstations.

Final version: December 16, 1994

To appear in proceedings of CHI'95

*Patrick Chiu is with LiveWorks, Inc., A Xerox Company, 2040 Fortune Drive, San Jose, CA 95131. Gordon Kurtenbach is now with Alias Research Inc., 110 Richmond Street East, Toronto, Canada M5C 1P1.

interaction enabled with such a representation is *freeform interaction*.[3]

Scribbling is a prime example of freeform interaction: In scribbling, strokes can be created (drawn) anywhere without affecting existing strokes. Any strokes can be changed or erased without affecting any other strokes.[4]

In contrast, traditional text editing is not freeform, because there is an underlying string structure among the characters; e.g., deleting a character affects the positions of all characters later in the string. To be freeform, characters would all have to be freeGOs, i.e., have no underlying string structure. Erasing some characters would not cause any other characters to move. Such a seemingly limited model of text would have a crucial advantage: characters and strokes could be freely intermixed.

IMPLICIT STRUCTURES

During freeform interaction, users create material on the display in order to react to it and see new relationships and to try new things with it by manipulating the material. There is — *in users' minds* — quite a bit of structure. The structure may be deliberate or it may emerge from the interaction. It may be partial and ill-formed. Because interaction is freeform, there is no constraint to remain within the conventions of a particular structure. There is often a mixture of structures on the display. Users' commitment to particular structures is often tentative and transitory, with the material being "re-registered" as different kinds of structures.

Such structure is *implicit* in the sense that it is perceived by the user but not by the system, because it is not defined or declared to the system. Keeping structure implicit is the essence of freeform interaction. The benefit is that users have the freedom to treat the material any way they want at any time; the cost is that the system cannot take advantage of the implicit structure in supporting the users' operations. Therefore, we would like the system to automatically perceive structure in the material in order to support a *WYPIWYG* (What You Perceive Is What You Get) capability [11]. But it is crucial to have the system perceive structure in the material only when the user needs support, and to keep the interaction freeform otherwise.

Our experience with LiveBoards and whiteboards is that list-like structures are ubiquitous. Thus, we set out to support the manipulation of four kinds of list-related structures:

- **lists** (vertically aligned sequences of items, which are horizontally clustered sets of GOs),
- **text** (horizontally aligned sequences of GOs),
- **tables** (arrays of elements, which are clustered GOs, aligned both horizontally and vertically),
- **outlines** (lists with indented items).[5]

The system need only perceive enough structure to support the "naturally expected" behavior of each of these structures.[6]

3. We are aware that the notion of freeform needs much more discussion. We present it here briefly to introduce a seemingly useful notion and to position the specific scheme described in the paper.

4. Strokes also have the property of being *freehand* (being able to take on arbitrary shapes based on the movement of the creating instrument). This is not to be confused with their being freeform (a property of the ensemble), which is the relevant property here.

5. Because of space limitations, we will say little about outlines in this paper. The design principles can be illustrated by our treatment of the first three structures.

6. "Full" recognition, such as handwriting recognition or "parsing" by a "visual grammar" (e.g., [6]), is not required.

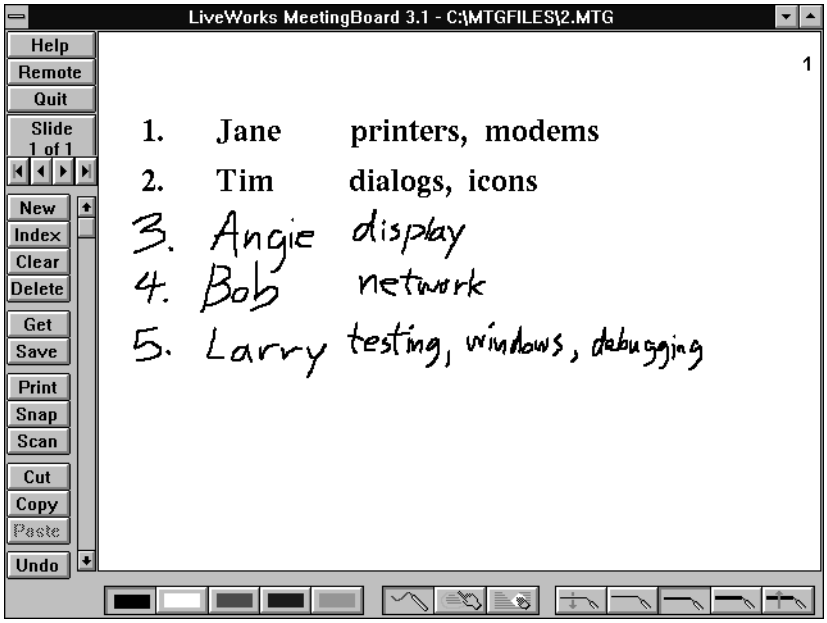

Figure 1. The commercial version of Tivoli: the Xerox LiveWorks MeetingBoard display.

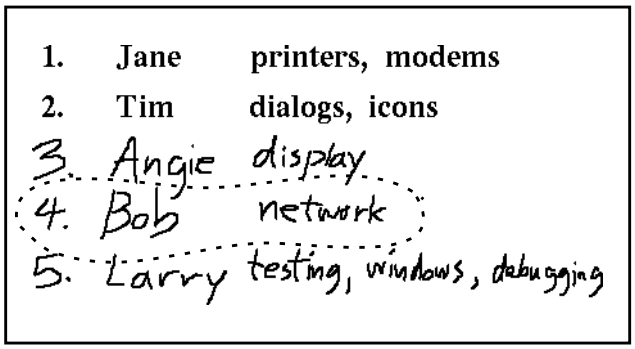

Figure 2a. Freeform move: User makes a loop gesture to select item 4, then drags the selection to before item 2.

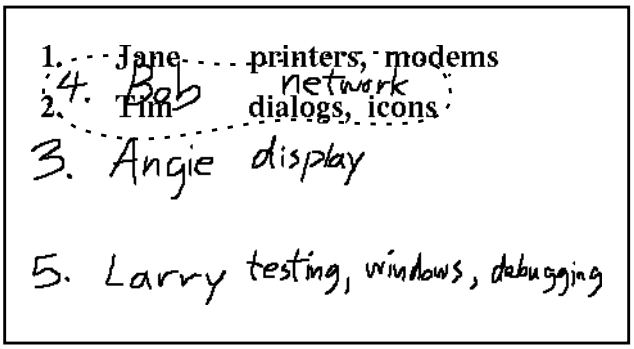

Figure 2b. Result: The selection remains exactly where the user left it, with no further adjustments.

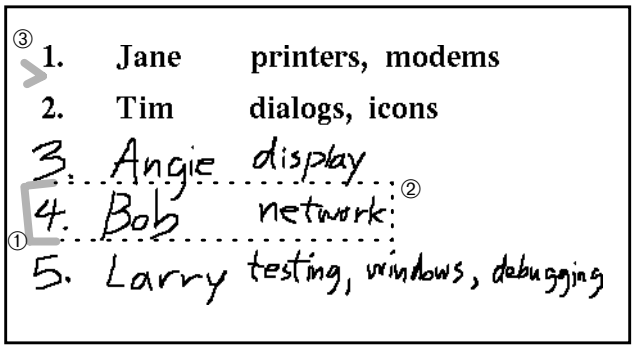

Figure 3a. Structured list move: User makes a left bracket gesture ①, which the system projects rightward into a structured selection ②. The wedge gesture ③ indicates the desired destination of the move.

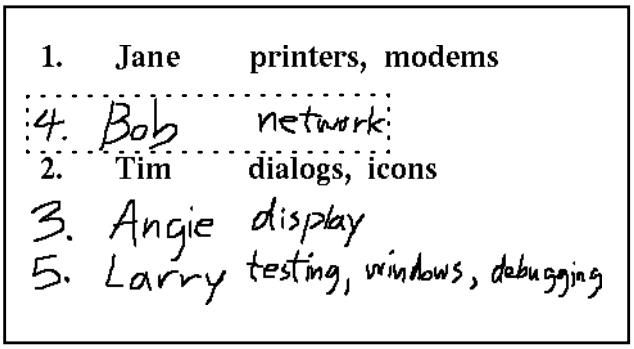

Figure 3b. Result: Items 2 and 3 are moved downward to open up space; item 4 is moved into the space.

The general design technique is to embed *ephemeral perceptual support* within freeform interaction: Whenever the user takes an action that implies a structural interpretation, the system temporarily perceives the structure in the material, carries out the current operation according to the expected behavior of that structure, and then returns to freeform interaction. Before discussing specific design techniques, we illustrate how our scheme works.

HOW THE BASIC SCHEME WORKS

Figure 1 shows the MeetingBoard display on the LiveBoard, which contains a mix of handwritten and typewritten material. The material is freeform, i.e., stroke and character freeGOs, but you can clearly perceive a lot of structure in it. It can be seen as a list of five items. Each item is numbered. After each number is a name. The names can be seen as a column. Horizontal clusters of characters and of strokes can be seen as words.

Let us see how to manipulate this material. Figure 2 shows a freeform move. First, a segment of material is selected by drawing a loop around it (Figure 2a); then it is dragged dynamically to a new location (Figure 2b). Because it is a freeform move, the dragged material stays where it is dropped, and none of the material around is adjusted. This is not satisfactory if the material is regarded as a list.

To deal with the material as a list (Figure 3a), the user first indicates the intent by using a structural selection gesture: a bracket ("[") gesture at the left of the list item. The system projects the legs of the bracket to the right to enclose the whole list item. The resulting rectangular selection enclosure signals that the system regards this as a structural selection. Then the user makes a wedge (">") gesture to tell the system where to insert the selected item. After the wedge gesture is made, the system opens up space for the item, moves the item, and closes the space where the item used to be (Figure 3b). The system animates all of the movement, so that the user, and the other people the user is working with, can easily track and understand the move.

To move a phrase, left and right bracket gestures are made to select the phrase (Figure 4a). Then a caret ("^") gesture is made to indicate a textual insertion. The system animates the moving of the phrase as well as the closing and opening up of spaces (Figure 4b).

To move a column, the material must be treated as a table. A top bracket indicates a column, the legs of the bracket projecting downward to select it (Figure 5a). A caret gesture shows where to move the column. Again, the system animates the move. All of the material to the right of the selected column is regarded as a column and moves left to close up the vacated space (Figure 5b).

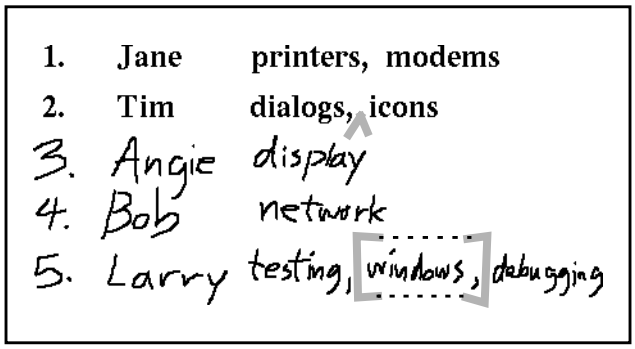

Figure 4a. Text move: User selects a word by making a left bracket gesture followed by a right bracket. The caret gesture indicates the desired destination of the move.

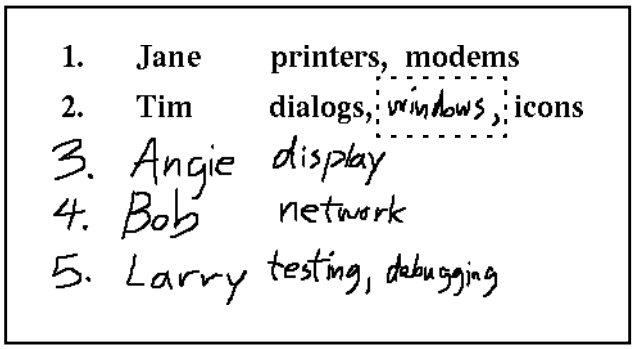

Figure 4b. Result: Space is opened up in item 2, the selected word is moved into the space, and the vacated space is closed up in item 5.

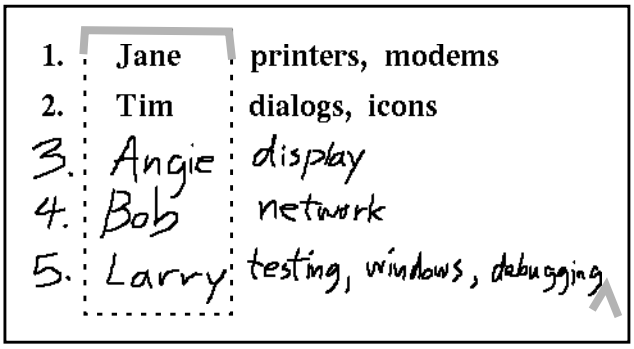

Figure 5a. Column move: User makes the "top bracket" gesture, then a caret gesture to indicate the destination of the move.

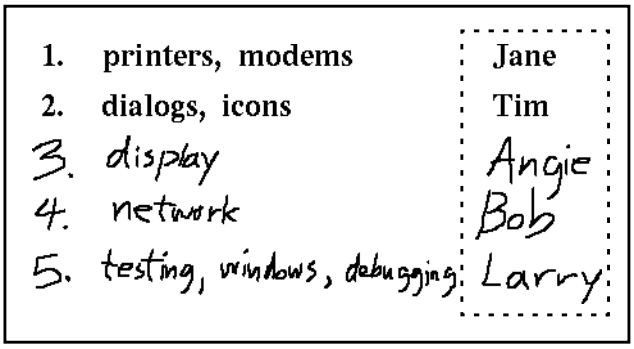

Figure 5b. Result: The selected column is moved to the right, and the space it vacated is closed up by moving the remaining objects leftward.

Any of the structural moves can also be made by dragging. To move a list item, the item is selected (as in Figure 3a) and dragged to a location near to where it is to be inserted (as in Figure 2b, except with a rectangular selection). Then the system animates the opening of space for the item, moving it from its dragged position, and closing the space where it started (the result being like Figure 3b).

Wedge and caret gestures indicate whether to insert a selection as an item or as text. In the case of dragging, the type of insertion is determined by where the selection is dragged to. If it is dragged to the gap between two lines, it is inserted as a list item; if it is dragged to a point within an item, then it is inserted as text.[7]

DISCUSSION OF DESIGN FEATURES

In developing the implicit structure scheme, we have been led to many unusual design decisions by our goal of working within a freeform interaction paradigm. Most of these decisions were arrived at during a process of iterating and exploring many alternatives. In this section we discuss some of the important features of the scheme.

Ephemeral Perception

The defining feature of implicit structure is that the system's perception is ephemeral — it is only in force for the duration of the immediate structural operations. The user interface issue is how to temporarily evoke the perceptual mechanisms in an effortless manner. Evoking structure is done by *gestural triggering* — making a structural selection gesture. The rectangular shape of the selection enclosure[8] makes it visually apparent that the system is ready to treat the next operation as a structural operation. After the user operates on a structural selection (e.g., moves it), the structural selection remains for further operations. The user can revert to freeform by simply beginning to draw strokes, and the selection is "dismissed" as a side effect.

We might argue that these transitional actions are costless for the user, since the user has to do them in any case. But, before making a selection the user must decide whether to make a structural or a freeform selection. The cost is mental (a choice has to be made [2]); and occasionally users make errors (e.g., selecting freeform but expecting a structural move to occur). We feel the benefits outweigh the costs.[9]

7. The drag "point" is the location of the stylus or cursor at the end of the drag.

8. In contrast, a freeform selection is enclosed by a freehand loop.

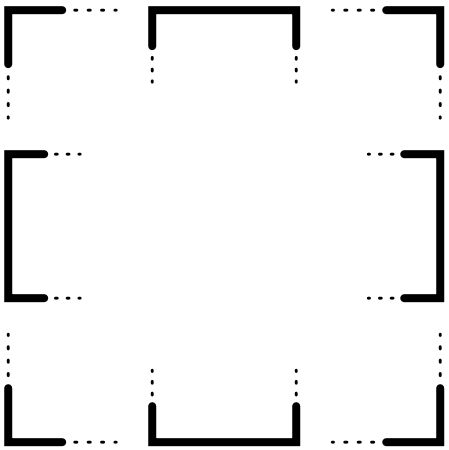

Figure 6. The eight structural selection gestures. Each gesture can be used to select the material enclosed by the projection of the legs, or to modify (extend or shrink) an existing structural selection.

Composite Structural Model

Each of the list-like structures we support has unique features; but they also share many common features, such as horizontal-vertical alignment, sequential elements, compactness (i.e., space preservation between items), and so on. Thus, we treat these four structures as a single *composite structural model*, rather than as four different models. There are two reasons: uniform selection and delayed commitment.

There is a uniform set of structural selection gestures, brackets and L-shaped gestures (Figure 6). These gestures work by projecting from their legs to define a rectangular region. This reduces structural selection to simple geometry, i.e., defining a rectangle. The user does not have to commit to a particular structure when a selection is made; the structure is not determined until an operation on the selection is invoked. For example, when an item in a list is selected, it can be either moved to another position in the list (with space opened vertically to make room for it) or to a place within another item (with space opened horizontally to make room for it).

Character-Based Structure

Characters are freeGOs in this scheme. Characters are created either by being imported from a text source or by being typed in.[10] In either case, they align nicely and look like structured text. But there is no underlying structure. If an eraser is swept through a neatly-aligned array of characters (such as those on this page), the characters touched by the eraser would be deleted, but none of the other characters would move; and the text-like appearance would be damaged.

Character freeGOs can be made to behave like text by invoking implicit structures. Structurally selecting a horizontal sequence of characters and then moving them will cause space to be opened and closed in a text-like manner. Typing is treated as an implicit structure operation. Making an ink dot on the display creates a type-in point; if the dot is near some characters, it will "snap" into a position so that the typed characters align with existing characters. If there are characters immediately to the right of the type-in point, they are moved to the right to accommodate the newly-typed characters.[11]

Generic Commands

Most of Tivoli's basic editing commands are generic (polymorphic); the same commands can be applied, with somewhat different but appropriate results, to different freeGO types and structures. For example, the same pigtail (delete) gesture applies to both strokes and characters, in either freeform or structural selections (in the freeform case the selected freeGOs are simply deleted, but in the latter case there is a further side-effect of moving surrounding freeGOs to close up the space). Erasing is always interpreted to be a freeform operation, and typing commands (character-creating and spacing commands) are always interpreted to be structural operations.

Borders & Multiple Structures

Consider a display in which a sketch is made to the right of a list. Structural operations are not possible on the list, because the system cannot perceive how much of the material on the display belongs to the list. If a left bracket gesture is made, it will project all the way to the right into the sketch. If the resulting "item" is then moved, the sketch will be altered in a nonsensical way (since it is not a list).

This situation is handled by the concept of *borders*. Very long strokes are considered to be borders that divide the display into regions. Borders delimit structures. Thus, structure operations stay within the confines of borders. In the exam-

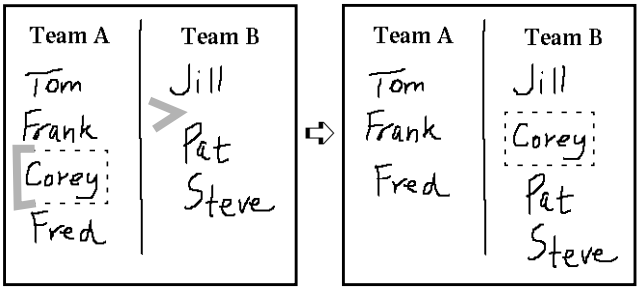

Figure 7. A "border" stroke confines the structural interpretation of selection and movement commands to one side of the border.

9. There is another important distinction for the user in Tivoli: The user must press a pen button to indicate that a stroke is a gesture. This sometimes causes errors. The benefit is that the user is totally free to draw anything without worrying whether some kinds of ink strokes are "reserved" as gestures.

10. Although it would be consistent with our scheme for characters to be recognized from handwriting, we do not support this, because it is not appropriate in the group work context of LiveBoard use.

11. Similarly, the keystrokes Space, Backspace, Tab, and Return evoke familiar positioning operations. There are, of course, no "formatting character" (e.g., Space, Tab, Return) freeGOs in this scheme.

ple above, if a vertical stroke is drawn between the list and the sketch, then operations on the list are confined to the left side of the border, and the sketch will not be disturbed.

The most common use of borders is to divide the display into columnar regions for multiple lists. Structural operations occur independently in the different regions. For example, an item can be moved from one list to another across borders. Figure 7 shows two lists with an item selected in the left list. When the user moves the item to the right list, it will be inserted there; and the resulting opening and closing of spaces will be confined by the border so they don't interfere with each other.[12]

Cleanup Operations

Structures emerge from freeform interaction. Because freeform interaction is not constrained, material is not always rendered precisely with respect to a given structure; and thus the system may not perceive it in the same way as the user does. This can lead to some unpleasant perplexities.

To deal with this problem, we provide *cleanup operations*, which neaten up the alignment of material. In carrying out a cleanup operation, the system must decide whether elements are aligned or not. It is not so important what the system decides; what is important is that it makes the system's perception clear to the user. The user can then adjust those elements that were misperceived.

For example, the horizontal cleanup operation is useful for tables. Consider the table in Figure 8a, which is taken from our user test data. The user created the table column-wise, and hence the rows are not well-aligned horizontally. Note that it is impossible to select the first row of the table because it dips and the elements are crowded. The horizontal cleanup operation analyzes a table column by column, identifying the items in each column, finds correspondences between the items in the different columns, and decides what is in each row; it then respaces the elements to make the spacing between rows clear. The result in this example is shown in Figure 8b, where it can be seen that the first row is now easily selected.

Animation

Structural operations, especially moves, cause not only the selected material to move but also some of the non-selected material. It can be confusing when many objects are jumping to new locations. This is especially critical in the LiveBoard situation, where one person is making the edits and other people are watching (the "demo phenomenon"). The operations on the display need to be *socially perceptible*. Our solution to this is to *animate* all the movements so that anyone watching can visually track the changes. (See [3,10] for a more detailed rationale for the utility of animation.)

Undo

The implicit structure scheme has evolved to be quite simple and natural. But this does not mean that anyone can walk up and use it without any practice. There are a lot of subtleties of interaction that must be learned through experience. Our experience is that it takes users about 10 minutes of practice to learn the basic Tivoli gestures plus about 15 minutes more to feel confident and comfortable with the implicit structure scheme. In LiveBoard situations, where users are in meetings when they first decide to use the structural operations, they cannot take time out to practice. Therefore, there is often a lot of initial fumbling. In these situations, we have found our unlimited *Undo operation* makes a huge difference. Even experienced users are occasionally surprised when the system perceives materials differently than they do (an inherent issue in this kind of technique), and the Undo reduces these potential disasters to manageable glitches.

THE SYSTEM'S PERCEPTUAL MECHANISM

The whole implicit structure scheme rests on the ability of the system to quickly perceive structure. The mechanism we use is based on computing a "projected ink profile." Figure 9 shows graphically the profile computed in both horizontal and vertical dimensions for the material from Figures 2 through 5. The peaks and valleys of the profile curves are analyzed to identify the structural elements and the spaces between them. While more sophisticated clustering techniques could also be used, this computational technique is

12. We also permit selections to be extended across borders. For example, in a table with vertical lines delimiting its columns, a "["gesture selects up to the nearest border. Then a "]" gesture on the other side of the border extends the selection across it. In carrying out the operation on the extended selection (such as moving a row of the table), the border lines are treated as "ambient" in that they remain fixed during the operation.

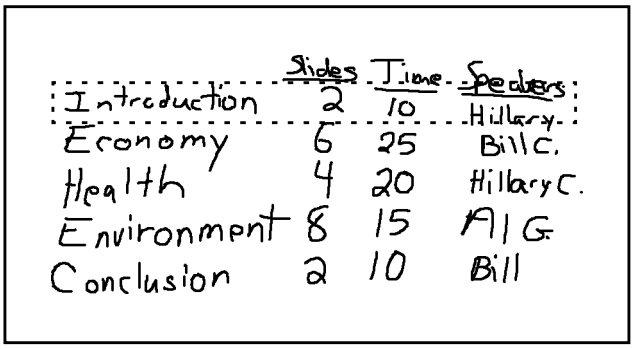

Figure 8a. This hand-drawn table has evolved to the point where the rows are too skewed and crowded to be selectable by a structured gesture.

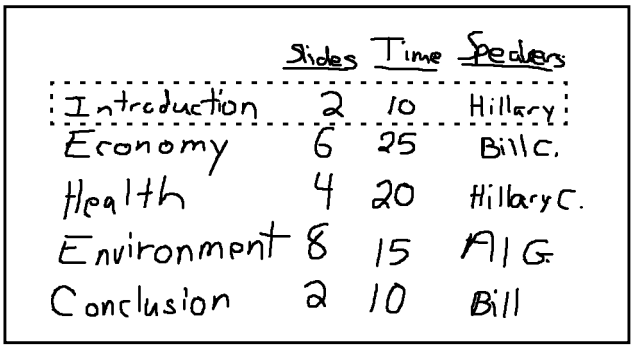

Figure 8b. After applying a "table cleanup" operation, the rows have been straightened out and spread apart, making them easily selectable.

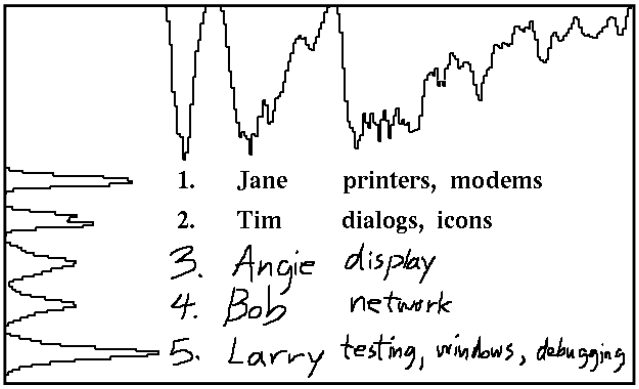

Figure 9. The projected-ink profile graphs at the left and top show the system's measure of the ink density in the x and y dimensions.

well suited to our purposes and has served us well. It is good at picking up the alignment relations that are inherent in list-like structures, while tolerating some degree of overlap between elements (e.g., the strokes in items 3 and 4 in Figure 9). It is efficient enough that structures can be computed on demand, which is critical for supporting ephemeral perception.

DESIGN HISTORY

Designing and iteratively refining the implicit structure scheme took place during a period of a little over two years. A variety of empirical tests were employed: experiences of the developers and close colleagues, independent user tests, and real use situations.

At first we implemented a "lined paper" method for handling lists (which is what other pen-based systems, such as [1,8], do). We set aside that scheme when we were satisfied that we could recognize lists on a blank surface. It took almost a year of iteration and refinement, with lots of testing within the Tivoli group before we had a version without "obvious" flaws. There were many design issues at this point that needed empirical evidence to help us address.

We conducted a small set of tests with independent users. In each session of about an hour, a user was trained until he/she was confident; then the user was given a range of tasks by playing the role of "scribe" in a simulated meeting. It took only four users before enough major problems were raised that we had to address. The problems involved confusion among the various structures (rows, columns, segments, blocks), which at that time were treated as different kinds of selection. During the next few months we redesigned and reimplemented the conceptual model and user interface: The structures were simplified to the composite structural model, and dragging and animation were added.

Another example of a problem raised in the user tests was that our early design for L-gestures was much too confusing to users. The original L-gestures are shown in Figure 10. The L-gestures were powerful operations for opening and closing spaces. Their design was perfectly logical: the order in which an L was drawn was significant; the first leg of an L indicated where the space was to be adjusted, and the second leg indicated the direction and extent of the adjustment. But users could not remember this abstract logic. Therefore, we abandoned these operations[13] and made all L-gestures be simple projective selection gestures (Figure 6). Users have no trouble with these.

Once these changes were made, we had an opportunity to use the system in support of a series of real meetings, whose function was to collaboratively reorganize the priorities among large sets of items. To do this they used the ability to manipulate multi-column lists. These meetings went on for several months, and prompted the refinement of several minor aspects of the interface and the workings of the underlying recognition algorithms.

Finally, a subset of the implicit structure scheme (most of the aspects described in this paper) was chosen to be "hardened" and incorporated into LiveWorks' MeetingBoard product.

RELATION TO OTHER SYSTEMS

There are hints of implicit structure in current user interfaces, although the principles are not articulated or developed. For example, most word processors implicitly recognize words, and yet allow users to deal with text at the level of characters. Emacs [14] carries this idea the farthest. Emacs uses the character string as the base representation (analogous to freeGOs), and allows modes to be applied to the base (e.g., text, Lisp, or C++ modes). The structures are embedded in recognition routines and not data structures, and thus different modes can be applied to the same text. Lakin took inspiration from Emacs and extended this idea to the visual realm [6]. But structure in Emacs is not ephemeral; the user must explicitly declare modes. Further, modes are applied wholistically to the entire character string.

Pen-based systems promise informal interaction. Yet they mostly use the pen to input characters and then treat the text in the standard way. That is to say, they are not freeform in the sense defined in this paper. Perhaps the most notorious pen-based system today is the Apple Newton MessagePad [8]. The Newton uses two basic structures, character strings and structured graphics. However, once handwriting or drawing is interpreted, the interpretation is permanent. Thus

Figure 10. A set of structured command gestures, no longer used in the present system.

13. There are other ways to open and close spaces.

handwritten text cannot be treated in a structural manner, and strings of characters generally cannot be freeform. Also, characters and graphics cannot be manipulated together in a structural manner; e.g., a graphic object located within a character string will not be moved when characters prior to it are deleted.

The a*h*a! InkWriter [1] is perhaps the closest system to ours in its basic objectives. Its main goal is to treat handwriting as text. It supports text paragraphs and lists of handwriting or characters. Graphics are treated as separate paragraphs. The system compromises on freeform interaction, because it uses a "lined paper" background. Strokes are interpreted as handwriting only if they occur between the lines, and strokes are interpreted as graphics only if they are more than two lines high (and at least one space from a handwritten paragraph). Once input is interpreted, it remains as either text or graphics. Neither system exhibits the fluidity or flexibility that we feel is necessary for a truly usable informal (i.e., freeform) system.

LIMITS OF THE IMPLICIT STRUCTURE APPROACH

We should put the notion of implicit structure within freeform interaction in perspective. Freeform interaction is appropriate only in situations where constraints would inhibit rather than support a process. There are times when constraints are helpful. We would not be against "freezing" (making explicit) an implicit structure in such a situation. What seems needed is a way to transition from freeform to structured interaction [12]. Treating structure implicitly and ephemerally is useful in early, formative stages of a process.

But it should be understood that there are limits to the implicit structure approach.[14] Given the inherent freedom of expression in freeform interaction, it is difficult for users to stay within the confines defined by particular "visual grammars." Even if users mentally stick to particular structures, there are the manual problems of neatness, in which users vary considerably. These make system perception difficult in general. That is why we have chosen not to implement elaborate recognition grammars, but rather to "perceive" simple visual features (e.g., alignment) that are useful across different structures.

We could mitigate the perceptual problems in several ways. The most heavy-handed way is a structure editor (like an Emacs mode). This is not acceptable. We could provide guidelines (like "lined paper"). This would be acceptable if the user were not confined in rigid ways to them. We have taken the "softest" approach: the use of cleanup operations.

In any case, this approach requires a spirit of cooperation between the user and the system. The users have to follow good-faith "interactional maxims" (analogous to the conversational maxims [5] that people naturally follow) if implicit structure is to work. We suggest a Maxim of Appropriateness, which says that users will only invoke operations that are appropriate given the material at hand (e.g., they will not try to do a list move on a sketch of a face). With experience, the user becomes more attuned to what to expect of the system (i.e., the sense of "appropriateness" becomes highly refined), and the interaction becomes skillful. The user can then have the benefits of structural support as well as freedom of expression.

14. In fact, these limits are probably inherent in all recognition-based systems.

ACKNOWLEDGMENTS

We would like to thank many colleagues — Eric Saund, Frank Halasz, Kim McCall, Steve Harrison, Sara Bly, and other members of the Collaborative Systems Area at PARC — for trying out and discussing this work as we iterated through various designs.

REFERENCES

[1] a*h*a*! InkWriter Handbook* (1993). Mountain View, CA: a*h*a! software corporation.

[2] Card, S. K., Moran, T. P., & Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.

[3] Chang, B. W., and Unger, D. (1993). Animation: from cartoons to the user interface. *Proceedings of UIST'93*, 45-55. New York: ACM.

[4] Elrod, S., Bruce, R., et al. (1992). LiveBoard: A large interactive display supporting group meetings, presentations and remote collaboration. *Proceedings of CHI'92*. New York: ACM.

[5] Grice, H. P. (1975). Logic and conversation. In P. Cole & J. Morgan (Eds.), *Syntax and semantics 3: speech acts*. New York: Academic Press.

[6] Lakin, F. (1987). Visual grammars for visual languages. *Proceedings of AAAI'87*, 683-688.

[7] Moran, T. P. (1993). Deformalizing computer and communication systems. Position Paper for the *InterCHI'93 Research Symposium.*

[8] *Newton MessagePad Handbook* (1993). Cupertino, CA: Apple Computer, Inc.

[9] Pedersen, E. R., McCall, K., Moran, T. P., & Halasz, F. G. (1993). Tivoli: An electronic whiteboard for informal workgroup meetings. *Proceedings of InterCHI'93*, 391-398. New York: ACM.

[10] Robertson, G. G., Card, S. K., MacKinlay, J. D. (1989). The cognitive co-processor architecture for interactive user interfaces. *Proceedings of UIST'89*. New York: ACM.

[11] Saund, E., & Moran, T. P. (1994). A perceptually-supported sketch editor. *Proceedings of UIST'94*. New York: ACM.

[12] Shipman, F. M., & Marshall, C. C. (1993). Formality considered harmful: experiences, emerging themes, and directions. Technical Report, Department of Computer Science, University of Colorado.

[13] Shipman, F. M., Marshall, C. C., & Moran, T. P. (1995). Finding and using implicit structure in human-organized spatial information layouts. *Proceedings of CHI'95*. New York: ACM.

[14] Stallman, R. (1985). *GNU Emacs Manual*. Cambridge, MA: Free Software Foundation.