Design Issues for Line-Driven Text Editing / Annotation Systems

Gary Hardock
Computer Systems Research Institute
University of Toronto
Toronto, Ontario
Canada M5S 1A4
gary@dgp.toronto.edu

Abstract

Recent research on interfaces driven by line-markings indicates that there are many potential benefits and applications of such interfaces. Benefits include the exploitation of users' handwriting skills and their skills in understanding handwritten marks. There are systems that have exploited one or the other of these benefits but not both. One application which would take advantage of both of these benefits is asynchronous collaborative text editing. In such an application, line-markings could be used for specifying commands, thereby taking advantage of users' handwriting skills, and for creating explanatory notes or annotations, thereby exploiting the users' understanding of handwritten marks. But there are many unknown and unsolved issues in designing such an application and for line-mark driven systems in general. This paper examines some of these issues in the context of an asynchronous collaborative text editing system. This system, termed MATE for Mark-up Annotator / Text Editor, is currently being designed and implemented.

Keywords: Annotation, Gesture, Line-Drive, Line-Marking, Mark-up, Text-Editor, Collaborative Writing.

1. Introduction

Recent research has shown that input in the form of line-markings,[1] such as characters, proofreaders' marks and other symbols, has many potential benefits in a wide variety of applications. These include spreadsheets, sketchpads, the entry of mathematical formulae and musical notation (Wolf, Rhyne & Ellozy, 1989), educational applications such as teaching writing skills and vocabulary (Chow & Kim, 1989), and text editing (Welbourn & Whitrow, 1988).

Wolf and Morrel-Samuels (1987) mention a number of potential benefits of using line-markings to specify commands:

- Line-markings can specify both a command and its arguments, often in a single motion.
- Temporal ordering of commands and syntactic information can be conveyed by the spatial form of the markings.
- Line-markings allow a more direct form of direct-manipulation than the "point and select" interfaces of mouse-based systems. Instead of first choosing an object and then a command, the command is specified directly on the object.

Another benefit of line-drive systems is that the exact placement and size of characters can be specified. For example, say we want to enter the following equations:

$$a_n = \sum_{i=1}^{n} k^i \qquad 2^3 \neq 3^2$$

It takes only about 10 seconds to write them out by hand, but over a minute to enter these using MathType and Microsoft Word on an Apple Macintosh. Much of this time is spent selecting menu items, various cursor positions and sections of text, typing on the keyboard, and most importantly, switching between these subtasks (see Buxton, 1990 pp. 13.5 for a detailed analysis). Time is only one aspect, the cognitive load of performing these subtasks is the major drawback of the point-and-select method. In contrast, the line-marking method doesn't present this cognitive load, so the task is both simpler and faster.

[1] In most of the previous literature on line-markings, the term *gesture* has been used to mean a "hand-drawn mark used to indicate a command and its arguments" (Wolf & Morrel-Samuels, 1987). This usage is misleading as the common definition of gesture is "an expressive movement of part of the body" (Oxford Paperback Dictionary, 1988). A gesture may leave a mark, but the mark itself is not a gesture. The hand and arm movements that created the mark could be considered gestures. The usage in this paper is consistent with what Buxton (1990) calls "line-driven" as opposed to "gestural" interfaces, which respond to the gesture itself rather than a line.

Line-markings are also useful for making *annotations*; notes of explanation. This is readily apparent as handwritten comments and marks are used in everyday life. Annotations are useful in collaborative work environments for communicating ideas amongst the people. There are several products already on the market and several research efforts that support collaborative annotations or markings. Products such as *Wang FreeStyle* allow one to mark-up a document and send it to other interested parties. The *Collaborative Annotator*, developed by Koszarek et al. (1990), also allows one to make annotations, but is menu-based and uses a mouse and keyboard for input, whereas *Wang FreeStyle* uses a stylus and graphics tablet. It should be pointed out that these systems allow multi-media annotations including voice, and that the *Collaborative Annotator* can also be used as a shared interactive tool.

A useful feature of line-mark annotations on text is what may be termed *figure-ground* distinction. That is, the type-written text tends to be viewed as background, while the line-markings stand out. This distinction is easily noticed in typewritten documents annotated with a pen. The benefit of this distinction is that the markings are easily distinguished from the text.

Line-markings are particularly useful in a collaborative environment. As mentioned above, they can be used as annotations for communicating amongst the members. They can also be interpreted by computer applications as commands. In the case of a text editor, these would be commands such as move, copy, insert and delete. The benefits are amplified when these two uses are merged into one system for the same markings can be used both as editor commands and as annotations.

The reason markings are preferred over other methods of input is that they are visible. That is, the entire command specified by a marking can be seen by the user. For example, one can view the history of editing operations performed on a document by viewing the marks used to edit it. This is not the case for typical direct manipulation interfaces as button clicks and keypresses are difficult to visualize. Keyboard-based interfaces can give command histories but are ill-suited for specifying locations, which is an integral part of most editing operations.

The visibility characteristic of markings permits actions or commands to be "deferred". Normally, the action specified by a mark occurs upon completion of the command. However, the action may be performed at any time after the mark is made because the mark can remain visible. It is this characteristic that allows markings to be treated as annotations until they are chosen to be applied as commands.

MATE (Mark-Up Annotator / Text Editor) is a first attempt to create a system which uses markings as both commands in text-editing and annotations in a collaborative writing environment. There are many design issues that must be examined to achieve this, and for many of these issues little or no literature exists. This paper describes the overall structure and design of MATE, examines some of the design issues and solutions, and describes the current implementation.

2. Overall Design of the Mark-Up Annotator / Text Editor

2.1 Overview

The MATE system serves as an *Annotator*, a *Text Editor*, and an *Annotation Viewer / Selector*. In *immediate* mode it is a Text Editor, as markings are immediately interpreted into editor commands and executed. In *deferred* mode it is an Annotator, for command execution is deferred or postponed and the markings stay on the document. In *view/select/edit* mode, MATE allows *deferred* markings to be viewed and selected for execution.

MATE is intended to support a group of people working together on writing a document. The general scenario is that there is a primary author who creates a document, either with MATE in text-edit (immediate) mode, or by using a compatible text-editor. This author then sends this document to the collaborators, who annotate it with changes and comments, using MATE in annotate (deferred) mode. They send the marked-up copies back to the primary author who can then select the annotations to perform as editor commands, and make additional changes with MATE in a combined view/select/edit mode.

The following is a brief description of the three modes of MATE and their integration. Detailed design issues are covered in the Design Issues, and Current Implementation sections.

2.2 Annotate / Deferred Mode

In *Annotate* or *deferred* mode, MATE serves two purposes. The first is to facilitate communication amongst users of the system; the second is to enter text-editing commands in a deferred mode which can later be viewed and executed. One of the primary benefits of the MATE system is that many of the annotations serve both purposes. For example, if I cross out a word (as shown in figure 1), I am communicating my intentions to any person who sees the annotation. This annotation could also be interpreted as an editing command, which can be executed when desired.

Some annotations can be used as both explanatory notes and editing commands, while others are only useful as explanatory notes. Examples of explanatory or comment annotations are comments such as "reword", "I think you should mention ...", etc., which do not correspond directly to any editing commands. These comments need not be recognized by the computer, as they are only intended to be understood by a collaborator.

There is no reason to restrict explanatory annotations to line-markings. One of the most useful methods of explanation is speech. Therefore voice recordings will be implemented into this system. This view is also held by many designers of annotation systems, including *Wang FreeStyle* and the *Collaborative Annotator*, mentioned above.

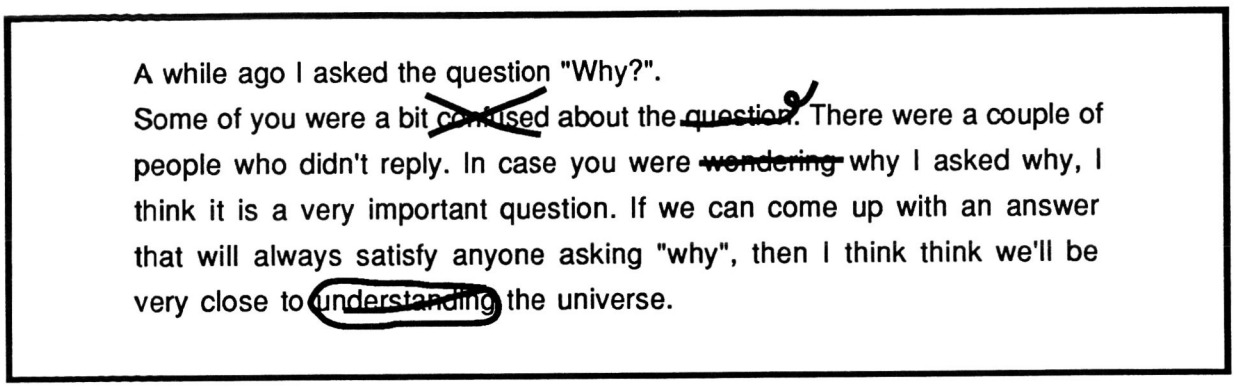

Figure 1 – Some possible deletion marks: cross-out, "pig-tail", stroke-out, and select & delete.

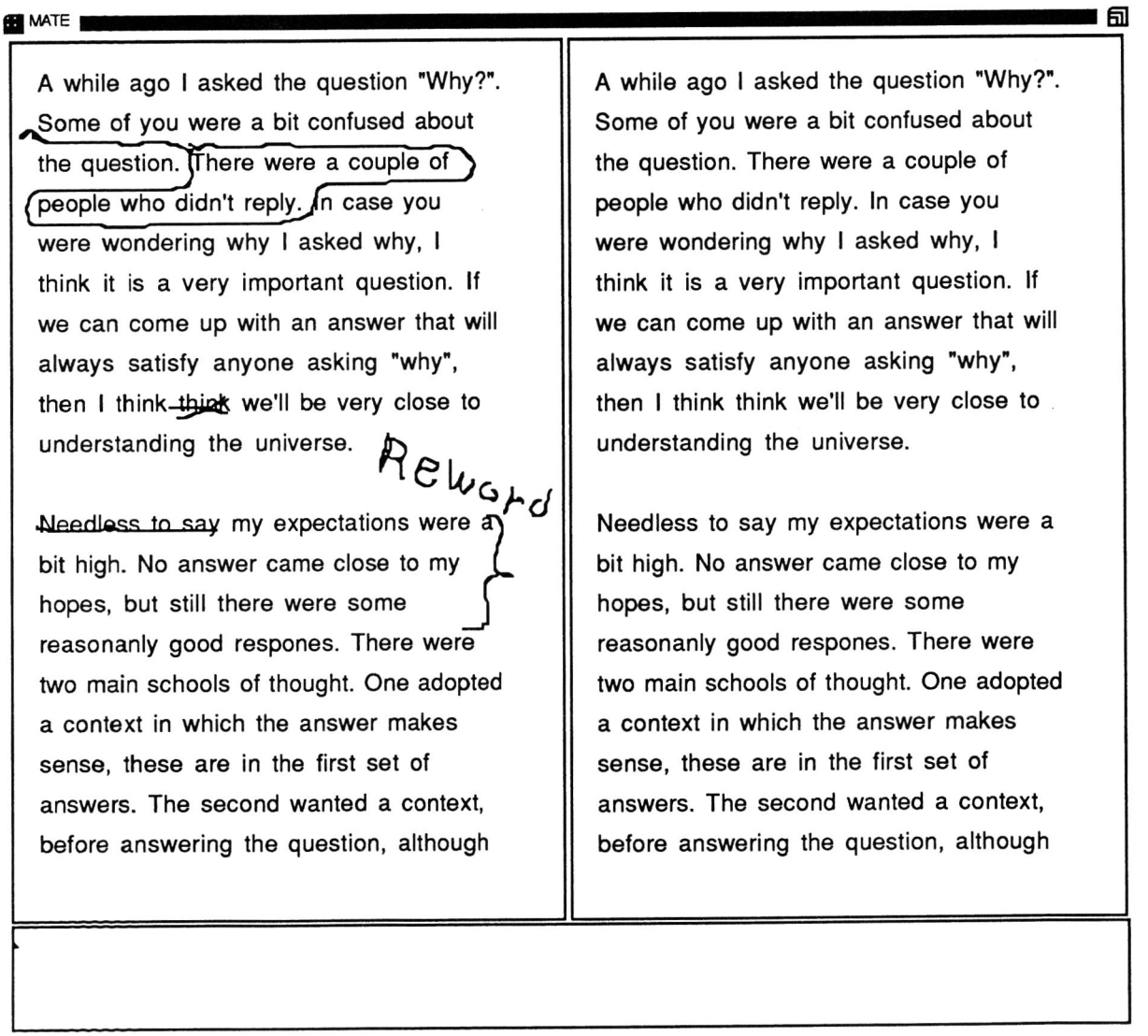

Figure 2 - MATE in View/Select/Edit mode

2.3 Edit / Immediate Mode

The immediate mode of MATE is different than the deferred mode for as soon as a line-marking is completed, it is interpreted and the corresponding editing command is performed. Mark-up or line-driven text editors have been investigated (Welbourn & Whitrow, 1988) and implemented (GO 1991a & 1991b), and are not the main focus of this paper. The design strategy for the immediate and deferred modes of MATE is that they are consistent with each other.

One issue that occurs due to this attempt at consistency is: what should be done with markings that are not recognized. This may cause problems depending upon the user's intentions. If the user intended a comment annotation to be included as part of the document, a major problem occurs. The text document is dynamic, i.e. it changes during the editing session. Determining what section of text an annotation applies to, if any, may become a difficult if not impossible task for both the user and the computer. This problem exists because annotations usually correspond to an area of a document. In an editing session, this area may be deleted, moved, or separated into several sections. Therefore comment annotations will not be supported in edit mode However, comments are supported in the view/select/edit mode of MATE, discussed in the next section.

The above problem does not apply to voice annotations for they can be placed at a specific location in the document as opposed to corresponding to a section of text. However, the first implementation of MATE will not support voice annotations in edit mode to avoid unnecessary complexity as voice annotations are not central to our current research.

2.4 View / Select / Edit Mode

This mode allows a user to *view* an annotated document, *select* annotations to be interpreted and performed as editing commands, and *edit* the document as in edit mode. The purpose of this mode is to support an author who may have annotated copies of a document from several collaborators, and wishes to incorporate these annotations along with additional changes. In order to accomplish this properly, the document is displayed in two windows, as shown in figure 2.

The right window is in Edit mode, with the additional feature that it can accept commands entered in the left window. The left window is in a mode similar to Annotate mode. One difference is that annotations can be selected to be interpreted as editing commands and performed on the document in the right window as if the command was entered in the right window. The other difference is that annotations from several sources may be displayed, using colour-codes to show the annotations' source. The integration of these two windows allows an author to incorporate suggested changes from any of a number of collaborators, or to ignore the suggestions and make his/her own editing changes.

3. Design Issues

The design issues discussed below have been separated into sections for presentation purposes; however it is important to note that no individual problem or solution is in complete isolation from the rest of the system.

3.1 Line-Marking Device

The important point about the line-marking device is that there are many small issues which could ruin an otherwise well-designed system. Computer applications that use line-markings as input can be assumed to use a stylus as the input device, in conjunction with either a tablet, flat-screen display or a regular display with a light pen. For the tasks of printing characters and drawing proofreaders' marks, other input devices, such as a touch tablet or mouse are poorly suited; touch tablets do not have the necessary resolution due to the size of the fingertip, and mice do not have the necessary accuracy due to limitations of the muscle groups used in controlling them.

However, there are many different types of styluses. Some can sense proximity, pressure and/or angle. Some have various switches or buttons attached to them. Some have tips that are stationary while other tips move under pressure. There are differences in the way the stylus feels to the user when it is held and moved across the sensing surface. All these factors have to be considered before choosing an appropriate input device.

Other hardware factors have to be taken into account as well. For example, Tappert et al. (1986) found that *parallax* was a major problem on a flat screen display. Parallax is the condition in which the ink-trail does not appear to the user to be aligned.with the stylus tip. He also noted other important factors such as the resolution of the surface, the sampling rate, and the agreement between the user and the system for when the stylus is "down" (touching the surface) and when it is "up".

It is important that a proper match is formed between the hardware and software. Even the metaphor used is important: the stylus might be considered a pen, pencil, highlighter, crayon, brush or a piece of chalk. Each metaphor brings with it certain expectations about the input device and the interface. For example extra pressure with a pencil is expected to make bolder lines whereas pressing down on a fine-tip marker or pen produces relatively little change in line quality.

The input device used in the current implementation meets the above criteria and is discussed in greater detail in the Current Status section.

3.2 Integrating Line-Markings into a Text Editing System

Line-markings may not be the best method for entering all of the commands that a complete system requires. There are two main types of commands for which the use of line-markings may be inappropriate; navigation commands, and general system commands such as those for file handling.

The pen and paper analogy gives the useful insight that one uses the non-dominant hand for auxiliary tasks, such as

turning pages, moving the paper around, and holding straight-edges and other devices for guiding the stylus. However, the pen and paper analogy does not apply to the general system commands. This is an important observation as the lack of analogy or a misapplication of the metaphor may give rise to major inconsistencies in the system.

Entering Navigation Commands, the Use of Touch Tablets and Two-Handed Input

There are two main approaches to navigate through a document. The first is to use a scrolling mechanism such as a scroll bar, or to "push" the cursor into the edge of the window to display more text. The second is by making discrete jumps, usually of a "page" in length.

There is actually an inconsistency in almost all editing systems used today, which disappears when line-driven input is used; there is no obvious cursor position after a navigation command is performed. An examination of a small sample of editors reveals major differences in the placement of the cursor after navigation. For example, editors on the Apple Macintosh keep the insertion point at the same location in the document, even if that part of the document does not appear in the window. The *vi* editor on *UNIX* places the cursor at the top of the window after a PageUp command and at the bottom after a PageDown command. Other editors have other variations. In contrast, with line-driven input, a cursor becomes unnecessary as positional information is given.

Using the pen and paper analogy, navigating by discrete jumps corresponds to the turning of pages. Touch tablets are very good input devices to use for this type of scrolling, as brushing one's finger against a touch tablet is very similar to turning the pages of a book. One benefit of touch tablets is that by using left-right motion to turn pages, forward-backward motion can be used to "scroll" through the page or document. Another benefit is that the non-dominant hand is still left free to hold rulers and other guidance devices.

The use of two-handed input for navigation/selection tasks has been studied by Buxton and Myers (1986), who showed that significant performance improvements can be made when both hands are used in such tasks. These results are transferable to the use of the touch tablet for entering navigation commands.

Entering General System Commands

The major difference between system commands and other editor commands is that they apply to the document or program as a whole, rather than to a particular part of it. In this case, the fact that positional data exist may cause confusion both to the user and to the system. There are many possible solutions, as the pen and paper analogy does not guide nor constrain the design of this part of the system. The most important point is to maintain the benefits and consistency of line-driven input. In particular, entering commands should not disrupt the user. What is meant by disruptive is that the continuity or flow of the user's actions is interrupted by large movements of the hand. For example, an *Undo* command located in a pull-down menu or a side button would violate the benefit of keeping the stylus near the area of interest by forcing the user to move to the menu or button and back again.

One solution is to use special symbols or characters to specify commands. This is very useful for commands with no arguments, such as *Quit*, *Undo*, and *Save File*. For commands with arguments the problem becomes greater. *Load File* and *Save to New File* not only require arguments, but may be required to assist the user in some way. For example, the Macintosh gives the user a scrollable list of files to choose from. In these cases the disruption caused by a scrollable list may be perfectly acceptable, as the commands themselves tend to cause disruptions — for example, loading a new file is usually expected to change the entire contents of the editing window. More importantly the nondisruptive commands such as *Undo* and *Save* should remain nondisruptive.

A second solution is to use a special area of the window and input area for entering these commands. This may be one of the better solutions for commands requiring arguments, but could be as disruptive as pull-down menus for commands which should remain nondisruptive.

A third solution is to use the non-dominant hand in some way. One method of using the non-dominant hand is to provide buttons for often used commands. The reasoning behind this is that the dominant hand has a lot to do already, whereas the non-dominant hand has only been given the navigation task. In the case of the *Undo* command, the two-handed solution has the added benefit that a command can be undone or canceled before it is completed thus taking advantage of the parallelism of two-handed input.

The above solutions are not mutually exclusive, each has its own advantages and disadvantages. It is possible to split the commands among the various methods, or to even allow commands to be entered in a variety of ways.

3.3 Commands That Cross Page Boundaries

A page is considered to be the section of a document which is displayed in the application's window. Some commands may need to cross page boundaries in order to specify their arguments. This includes all commands that require a section of text to be specified (e.g. Move, Delete, and Copy), and/or require a destination (e.g. Move, and Copy). The first situation, specifying a section of text across page boundaries, is called the *Disjoint Scope Specification Problem*, and the second situation, specifying a destination located on a different page, is called the *Remote Destination Problem*. These are slightly different problems and are dealt with separately below.

The Disjoint Scope Specification Problem

There are numerous methods to specify scope (i.e. sections of text): circling, bracketing, and highlighting are three of the most popular. *Circling* means to surround the scope with a closed loop; *bracketing* means to specify the start and end points of the scope separately, usually with symbols resembling brackets; and *highlighting* is similar to dragging through text with a mouse. One problem with highlighting is that it could be confused with the Deletion or Underline markings. Circling has the dual benefit that it

is specified in one continuous motion, and that both the system and user can be in agreement that the scope has been specified (i.e. once the loop is connected both the system and user know that the scoping part of the command is complete). Bracketing has the disadvantage that it requires more than one continuous mark. This leads to the "Dangling Brackets Problem". This is the problem that the system is expecting a second bracket to be entered, but the user might enter intermediate markings and commands, and may even forget about the first bracket. The dangling bracket problem also occurs when a user is viewing annotations, for the user must attempt to match pairs of brackets.

Scoping across pages and the dangling brackets problem are part of a larger problem which Rhyne (1987) terms *Embedded Dialogues*. This occurs when a partially specified command is temporarily interrupted while a sub-dialogue such as navigation is performed. Once the sub-dialogue is finished, the user finishes entering the command. Figure 3 shows an example of this in which a user has navigated down one page before finishing the command. Rhyne points out that this is a very problematic part of line-driven input, mainly because embedded dialogues are very difficult for the computer to decipher. Embedded dialogues also cause problems for the user. The user may forget about being in a sub-dialogue, or which sub-dialogue he/she is currently in. For a user viewing annotations created with embedded dialogues the problem is even worse, as the user would need to decipher how the annotations were made.

One solution is to think of the system as a command parser with the interpreted line-markings as the input to the parser. Some line-markings could be interpreted as partially specified scopes such as top-half or bottom-half, left-bracket or right-bracket. If the next line-marking is not the other half of the partially specified scope, the parse will fail and the command will be rejected. However, problems may still exist, for a user viewing annotations would still need to match bracket pairs.

The problems of embedded dialogues might disappear with the use of a separate input device for navigation. The input becomes more of a parallel dialogue, and it may be possible to avoid the need for embedded dialogues along with the problems associated with them.

The Remote Destination Problem

This problem is simpler than the scoping problem as a destination corresponds to a single point rather than to an area. Here we can borrow a useful technique used with the pen and paper analogy. When specifying a destination on a different page, one often makes a mark such as an asterisk or a circled number or letter as a temporary "destination placeholder". On the page containing the actual destination, the placeholder is treated as the source, and the actual destination is specified normally. Figure 4 shows an example of this procedure.

This method has many uses other than the one for specifying distant destinations. It can be considered as a placeholder serving a similar function to that of the Macintosh clipboard. In contrast, any number of placeholders can be specified, whereas the Macintosh only supports one clipboard. Placeholders can also be used as markers to certain locations in the document for future reference. For example, a command to go to the page containing a certain placeholder would be useful.

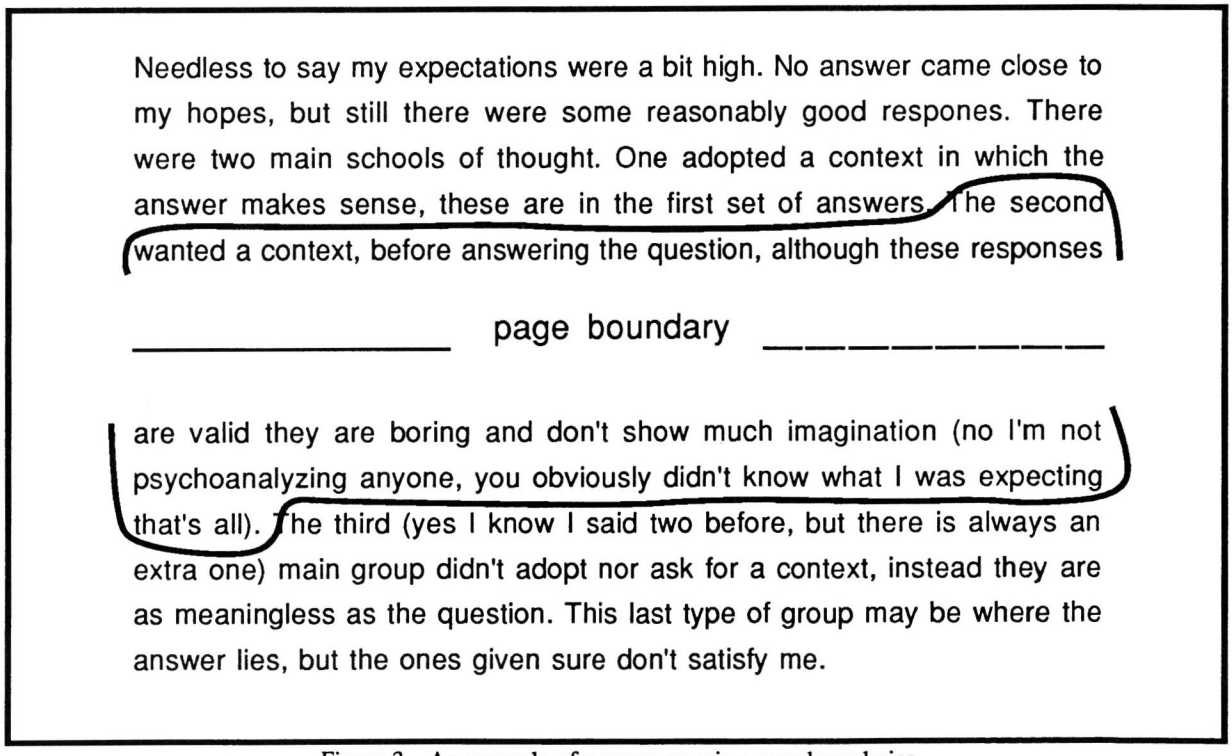

Figure 3 - An example of a scope crossing page boundaries

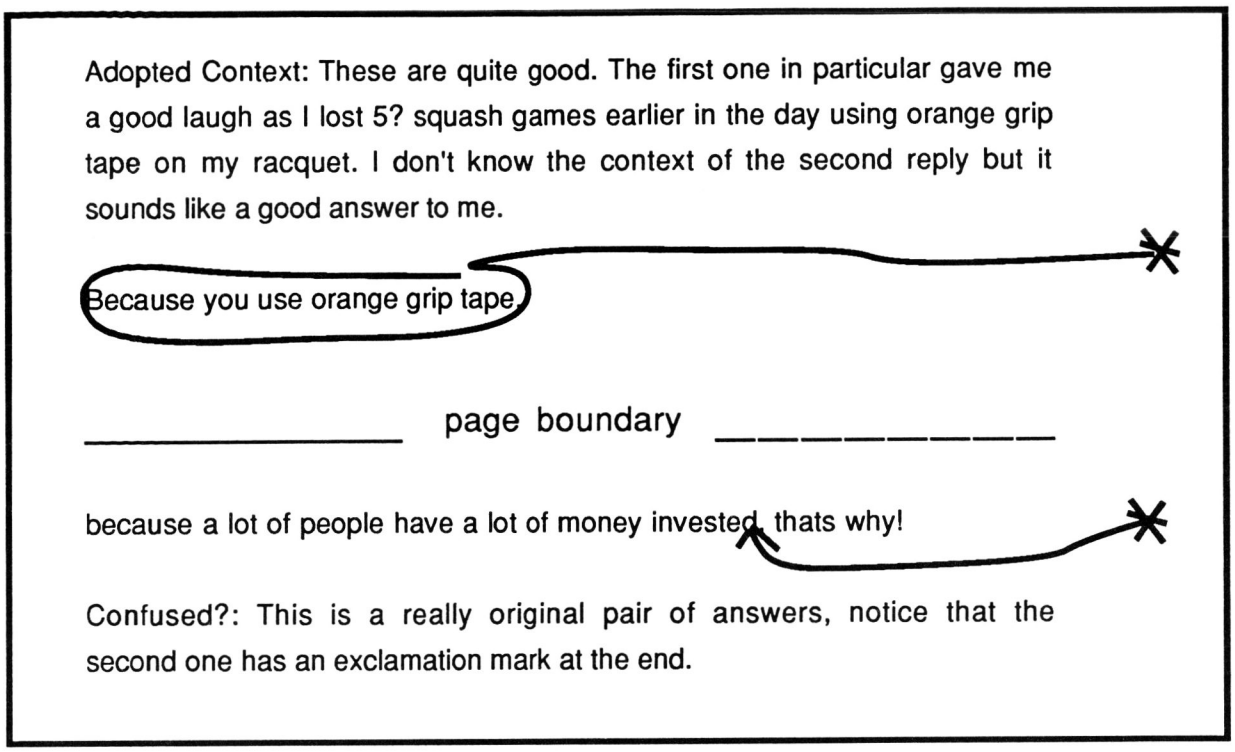

Figure 4 - An example of specifying a remote destination using placeholders

4. Current Status

MATE is currently being implemented with a tablet and stylus made by Pencept, and a separate display using the X windowing system on a Sun workstation. A more ideal system would employ a flat-screen display on which the stylus is directly used. The hardware selected for system implementation was chosen due to its availability and may affect user performance and satisfaction. However, the tablet and stylus being used are satisfactory according to Tappert's (1986) findings.

Working versions of Annotation mode and Edit mode exist, but no formal user testing has been performed. Instead an informal study has been carried out. Copies of an earlier draft of this paper were given to several proofreaders, each of whom was asked to mark up the paper using a coloured marker on transparencies laid over the paper. A preliminary analysis shows that the markings become very cluttered and difficult to see, and that there are many conflicts among the proofreaders' annotations.

The current version of MATE has borrowed the Move, Copy, and Delete commands from GEdit (Kurtenbach & Buxton 1990). These commands were intended for graphical objects, but little modification was required to adapt them for text. Future work may involve using more appropriate markings for text editing.

5. Summary and Future Work

MATE is being designed and developed to incorporate the ideas discussed in this paper. There are many unsolved and unknown issues in the use of line-mark text-editing, asychronous collaborative writing and line-mark systems in general. The current emphasis of this project is to complete the system by supporting the selection of annotations or deferred editing markings. Once this is done, MATE will serve as a testbed to further explore these issues and design problems, as well as bring to light new issues. It will also serve as a preliminary means for comparing line-mark based systems versus other systems that perform similar functions, such as annotating documents, text-editing, and asynchonous collaborative text-editing.

This paper is intended to provide an overview of the MATE project, discuss some of the issues involved in designing such a system, and shed useful insights for those working in this field.

Acknowledgements

I would like to acknowledge the members of the Input Research Group at the University of Toronto who provided the forum for discussing and contributing to the work presented in this paper. In particular, I would like to thank William Buxton and Gordon Kurtenbach who provided many useful insights, ideas, and the recognition software from GEdit.

References

Buxton, W. & Myers, B. (1986). A Study in Two-Handed Input, *Proceedings of CHI'86*, pp. 321 - 326.

Buxton, W. (1990). The Pragmatics of Haptic Input, *Tutorial Notes, CHI'90*, Seattle, Washington.

Chow, D & Kim, J. (1989). Paper-Like Interface for Educational Applications, *National Educational Computing Conference '89*, Boston, Massachusetts, pp. 337 - 344.

GO corp. (1991a). The Point of the Pen. *Byte*, February, pp. 211 - 221.

GO corp. (1991b). Video Presentation at Stanford University, by Robert M. Carr, February 13.

Kosarek, J.L., Lindstrom, T.L., Ensor, J.R. & Ahuja, S.R. (1990). A Multi-User Document Review Tool, in S. Gibbs & A.A. Verrijn-Stuart (Eds.), Multi-User Interfaces and Applications. North-Holland, Elsevier Science Publishers, pp. 207 - 215.

Kurtenbach, G. & Buxton, W. (1990). GEdit: a testbed for editing by contiguous gesture. To appear in the *SIGCHI Bulletin*, April, 1991.

Rhyne, J.R. (1987). Dialogue Management for Gestural Interfaces. *ACM Computer Graphics* 21-2, pp. 137 - 142.

Tappert, C.C., Fox, A.S., Kim, J., Levy, S.E. & Zimmerman, L.L. (1986). Handwriting Recognition on Transparent Tablet Over Flat Display, IBM Technical Report RC 11856 (52695) 3/3/86, also in Society for Information Display, Digest of Technical Papers, vol 17, San Diego, pp. 308 - 312.

Welbourn, L.K. & Whitrow, R.J. (1988). A Gesture Based Text Editor, in D. Jones & R. Winder (Eds.), People and Computers IV, Proceedings of the Fourth Conference of the British Computer Society Human-Computer Interaction Specialist Group. Cambridge, Cambridge University Press, pp. 363 - 371.

Wolf, C.G. & Morrel-Samuels, P. (1987). The use of hand-drawn gestures for text editing, *International Journal of Man-Machine Studies*, 27, pp. 91 - 102.

Wolf, C.G., Rhyne, J.R. & Ellozy, H.A. (1989). The Paper-Like Interface, IBM Technical Report RC 14615 (64399) 2/3/89, also in Designing on Using Human Computer Interfaces and Knowledge-Based Systems, G. Salvendy & M.J. Smith (Eds.), Elsevier Science Publ, Amsterdam, 1989, pp. 494 - 501.