# A MARKING-BASED TEXT EDITING SYSTEM FOR COLLABORATIVE WRITING

GARY JAMES HARDOCK

1993

# A Marking-Based Text Editing System for Collaborative Writing

by

*Gary James Hardock*

Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
M5S 1A4

A Thesis submitted in conformity with the requirements for the Degree of Master of Science in the University of Toronto

# Abstract

Markings have been used to communicate to people, in the form of markup annotations, and to communicate to computers, via a machine-understandable marking language, but these two uses have never been integrated. In order to investigate such an integration of these two uses of markings, we designed, implemented, and user tested a marking based collaborative text editing system termed MATE for Markup Annotator / Text Editor. This exploratory research can be applied to three areas: asynchronous collaborative writing, the visibility of markings, and interaction languages which can be understood by both people and computers. In addition to these research topics, we are also interested how each topic interacts with the others.

Through the design, implementation, and usability testing of MATE, many insights were made:

- The properties of markings, visibility in particular, have not been fully exploited and allow marks to be used in many new ways.
- Common interaction languages are very useful even when the computer can only understand a limited subset of the language.
- Asynchronous collaborative writing can benefit in many ways from the integration of annotations and editing commands.

Rather than studying each research area in isolation, we concentrated on a specific application which enabled us to find new insights in each area from our knowledge of the other areas.

# Acknowledgments

I would not have been able to finish this thesis without the help of many people. I would like to take this opportunity to thank them:

- Bill Buxton, my supervisor, for suggesting this thesis topic, keeping me motivated, and providing me with countless ideas;
- Mark Chignell, my second reader, for his many helpful comments and for greatly improving this thesis;
- Gordon Kurtenbach for being like a second supervisor to me;
- Marilyn Mantei for introducing me to the field of Human-Computer Interaction;
- the Input Research Group for providing the research environment within which this thesis was undertaken;
- the DGP system administration staff: Dimitrios Nastos, George Dretakis, and Mike Bonnell;
- all the people in DGP for the friendships and interesting discussions, especially Gord, Dimitrios, George, Beverly, Chris, George, and Kelly;
- Kimeda Sensei, my Aikido instructor, and David A. Burt, my piano teacher, for all they have taught me, and for helping me maintain a balanced life;
- Mihaly Csikszentmihalyi for writing "Flow: The Psychology of Optimal Experience" (Csikszentmihalyi, 1991);
- Neil A. Fiore for writing "The Now Habit" (Fiore, 1989) which helped me overcome my procrastination habits (and, yes, I did finish reading it);
- Marvin Theimer and the people at Xerox PARC for an enriching summer;
- Digital Equipment Corp., Xerox PARC, and the Natural Sciences and Engineering Research Council of Canada for their financial support;
- Dimitra Vista for her special friendship, for her many helpful comments on the early drafts, and for discussing what "My thesis is ...";
- LouAnne Johnson for more than I could even begin to write about here, but especially for being herself;
- Koula Bouloukos for more than I could ever put into words, may we never stop learning;
- my dog Chloe for never asking me when my thesis would be done;
- my family for all their love.

My appreciation for all of you goes far beyond what I have written here.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Pen-based computing is now becoming a reality as pen-based computers, operating systems, and applications are being developed. Much work has been done in many areas of pen-based computing; our particular interest is text editing. Several research and development efforts have examined pen-based text editing (Carr, 1991; Goodisman, 1991; Welbourn and Whitrow, 1988; Wolf and Morrel-Samuels, 1987), but we are also interested in another aspect of the writing process – the annotating of documents. People mark-up documents everyday to communicate with each other. This communication is supported by applications such as Wang *FreeStyle* (Francik and Akagi, 1989; Levine and Ehrlich, in press; Perkins, et al., 1989), which have been built to support the marking-up of documents electronically.

Our motivation is that there is a gap in all this work; as these two uses of markings – for specifying text editing commands, and for annotating documents – have never been combined in an integrated manner. The goal of this thesis is to fill this gap by designing and building a marking-based text editing system which allows users to annotate text documents, edit text documents, and to incorporate annotations as editing commands, thus integrating the two uses of markings. This system is termed MATE for Markup Annotator / Text Editor.

The gap in previous work not only exists in the research of markings, but also in the text editing process. People use mark-up annotations to communicate with each other in collaborative writing efforts, but to actually incorporate the changes they must manually convert the annotations into text editing commands, which they then have to enter manually. By integrating the two uses, the computer will be able to understand some of the annotations, and thus reduce the need for manually converting and entering the commands.

Our first research question is how can the integration of the two uses of markings be achieved. Our answer comes from the observation that marks used for annotations rely on the fact that marks are visible, whereas visibility is not exploited in marks used for specifying commands. By keeping the marks intended as commands visible, they can also be used as annotations.

Along with investigating the integration of the uses of markings, our research can be applied to the following three areas:

- *Asynchronous collaborative writing:* This refers to those stages in a collaborative writing process, in which the members work separately, and thus use annotations to communicate with each other. Rather than building a system which just integrates the uses of markings, we use the application of asynchronous collaborative writing to guide the design and implementation of MATE. This gives us a dual benefit, in that we are researching the application of asynchronous collaborative writing by building a system which supports it, and we can gain many insights into the integration of the uses of markings as many of our initial insights came from this application. A brief description of the asynchronous writing scenario for which we believe MATE to be the most suited for is described in the next section.
- *The visibility of markings:* As mentioned above, the fact that marks are visible allows them to be used as annotations, and this property has not been exploited in the use of marks as commands. Our research will allow us to identify and examine the benefits and constraints when visibility becomes an important consideration for marks used as commands.
- *Common interaction language:* Most interactions with a computer are intended solely for communication with the computer. By using the same marks as both annotations and commands, a common language understood by both people and computers results. A key point of such a language is that the user is required to learn either nothing new or very little; the computer must learn the user's language. This concept is the main thrust behind Natural Language Understanding research, but our marking-based approach allows us to investigate this topic from a different perspective.

Additionally, the interaction between each of the above research areas is important to investigate. Many insights in one area can be found from observations in the other areas. This was already shown in the dual benefit of designing and implementing MATE for the application of asynchronous collaborative writing. Another example is that asynchronous collaborative writing can be improved if an editing language common to both people and computers is used, and a common interaction language can be achieved by using proofreaders' marks, found in asynchronous collaborative writing.

This exploratory research is carried out by the design, implementation, and user testing of a prototype system, termed MATE for Markup Annotator / Text Editor. Through this process we intend to gain insights into research areas mentioned above, as well as the interactions among them. Because these areas are intertwined, they are covered in parallel throughout this thesis.

In the remainder of this chapter, we describe the asynchronous collaborative writing scenario of interest to us, and the general design of MATE. Our use of markings binds the three areas of research; therefore we examine markings in detail in Chapter 2. Chapters 3, 4, and 5 describe the design, implementation, and user testing of MATE, respectively. Finally in Chapter 6 we come back to our research areas and apply the insights gained from MATE to these.

## 1.1 Asynchronous Collaborative Writing

Almost everything written today is accomplished through a collaborative effort. Many works are written by several authors, but even if there is a single author, proofreaders and editors make comments and changes to the document before it is complete. This collaborative writing process contains several stages: brainstorming, researching, planning, writing, editing and reviewing (Posner, 1991). Some stages, such as brainstorming, are more well suited to group meetings, whereas other stages, such as editing, are performed individually and then presented to the group or to the person in charge of the document. We are concerned more with the later stages of editing and reviewing, in which the collaborators tend to work separately on making changes which are then later communicated to the rest of the group. Such stages, in which the members work separately, are termed *asynchronous collaborative writing*.

The collaborative effort can be organized in several ways (Posner, 1991). If there is a sole author then that author is in control of making any changes to the document. In multiple author scenarios, the ownership or control might be given to a single author, divided into sections or chapters, or subject to group consensus. The scenario of interest in this thesis is that the document is under the control of one person. In this scenario, the document moves through the following process, shown in Figure 1.1:

- The principal author creates a draft of the document, and then sends copies to the other collaborators.
- These collaborators review the document, annotate it, and then return the annotated document to the principal author.
- The principal author then incorporates the suggested changes into the document.

Currently, the above process is accomplished by sending paper copies of the document back and forth, and by marking up these paper copies with a pen. Even if both the principal author and collaborators have access to electronic mail and word processors, they will more than likely use a paper copy at some point in the process, mainly for the purpose of communicating their suggestions to another collaborator.

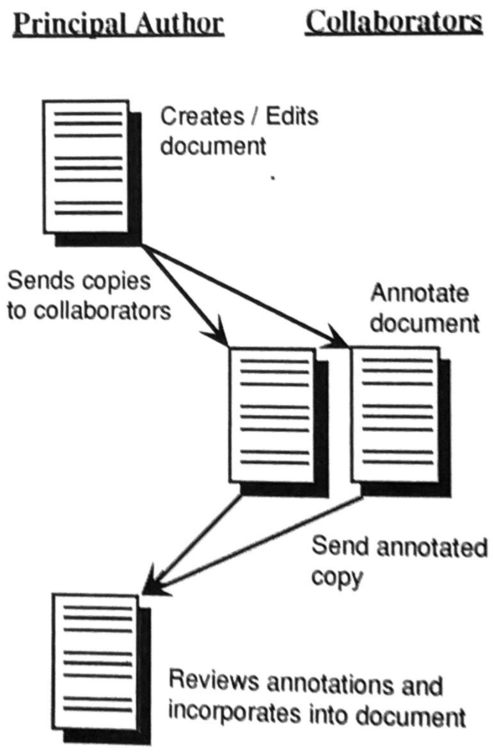


**Figure 1.1:** The Asynchronous Collaborative Writing Scenario
*The principal author creates a document and then sends copies of it to collaborators. The collaborators then annotate their copies and send them back to the principal author. The principal author then reviews the suggestions and possibly incorporates them in the newer version of the document.*

Wang *FreeStyle* allows its users to do all their handling of a document electronically. This is accomplished by creating a bit mapped image which can be transmitted electronically, and annotated with pen and voice. But the bit mapped image is only an electronic version of paper; a computer document could and should have many more advantages. By building a system which integrates mark recognition with annotations, we intend to improve upon the existing collaborative writing process.

## 1.2 MATE (Markup Annotator / Text Editor) – General Design

In this section, we provide a general description of how MATE is designed. A detailed description of MATE's design and the reasons behind the design decisions is given in Chapter 3.

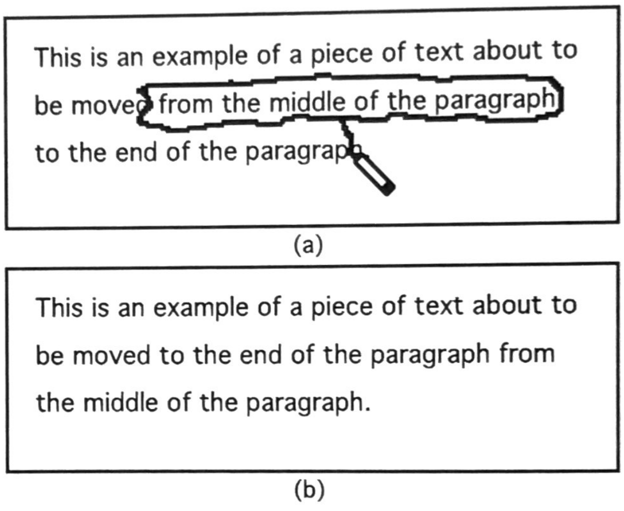
In order to integrate the uses of markings and to support our collaborative writing scenario, MATE allows users to:

- Edit text documents
- Annotate electronic text documents
- Incorporate the annotations into the new version of the document.

These three functions map into three modes, each supporting one of the above uses.

- To edit text documents, MATE acts as a mark-based text editor (Figure 1.2), such as (Carr, 1991; Goodisman, 1991; Welbourn and Whitrow, 1988; Wolf and Morrel-Samuels, 1987).
- To annotate text documents, MATE acts as electronic paper (Figure 1.3), like Wang *FreeStyle*.
- To incorporate annotations into the document, MATE uses two views of the document (Figure 1.4). The left view is similar to Annotation Mode, the right view to Edit Mode. The main difference is that the user can select a mark in the Annotation View which is then interpreted and executed as an editor command, with the result shown in the Edit View. The design of MATE is described in more detail in Chapter 3 – Design.

MATE is not intended to be the best annotation tool or text editor, but rather a first attempt at a system which integrates both uses. We are mainly interested in the user interface issues, as it should first be determined how usable such a system is before dealing with other issues. But other issues, such as how the underlying data structures are handled, are important and are also dealt with.

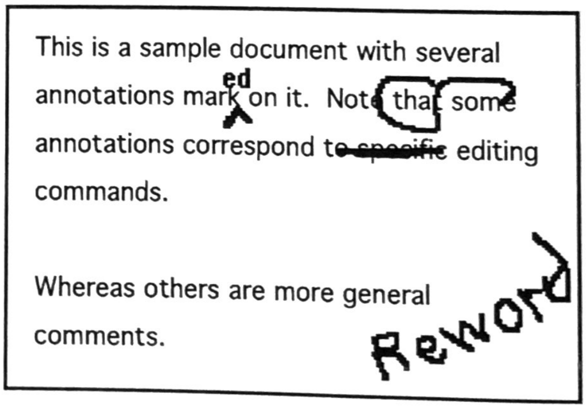

This is an example of a piece of text about to be moved from the middle of the paragraph to the end of the paragraph.

(a)

This is an example of a piece of text about to be moved to the end of the paragraph from the middle of the paragraph.

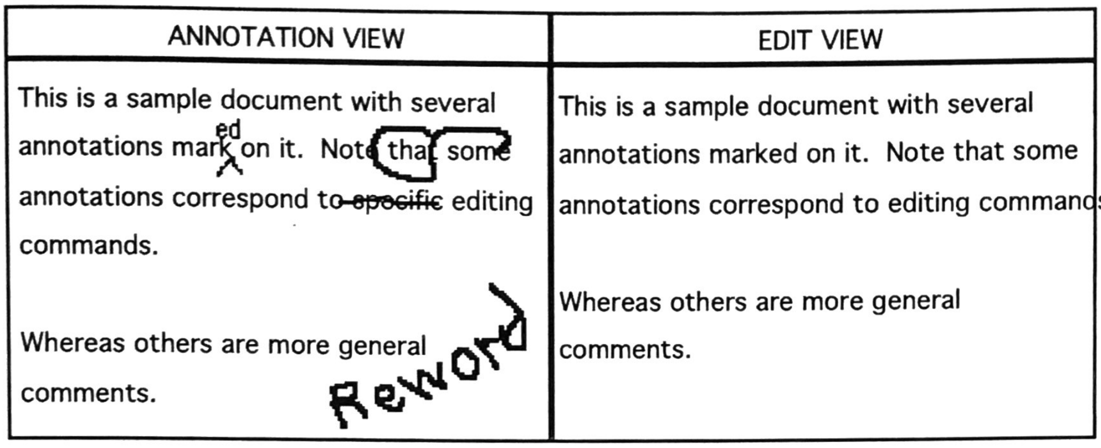
(b)

**Figure 1.2:** MATE in Edit Mode
*(a) A user draws a move command. (b) After lifting the pen, the editing command is performed, and the mark disappears.*


This is a sample document with several annotations mark on it. Note that some annotations correspond to ~~specific~~ editing commands.

Whereas others are more general comments.

**Figure 1.3:** MATE in Annotation Mode
*In annotation mode, users mark up a document in MATE just as if they were marking up a paper document.*



| ANNOTATION VIEW | EDIT VIEW |
|---|---|
| This is a sample document with several annotations mar$\overset{ed}{k}$ on it. Note that some annotations correspond to ~~specific~~ editing commands.<br><br>Whereas others are more general comments. Reword | This is a sample document with several annotations marked on it. Note that some annotations correspond to editing comman<br><br>Whereas others are more general comments. |

**Figure 1.4:** MATE in Incorporation Mode
*In incorporation mode, a user can view the annotated document, and select which annotations to incorporate. Annotations t(seen in the left window) that have been "executed" appear as thin lines (e.g.., "ed"). Annotations that have not been executed appear as thick lines. Annotations are colour coded according to who made them. Annotations that represent commands can be executed by selecting them with the stylus. Annotations that have been executed can be "undone" by selecting them. The current state of the document appears in the right hand window. The user can navigate (scroll) independently or synchronously in each window.*

By building MATE we intend to do the following:

- Gain an understanding of how the integration can be actually achieved
- Gain insights into the benefits of the integration
- Determine the implications of such a system in the design and implementation of both the user interface and the rest of the system.
- Test the usability of the concepts developed

## 1.3 Summary

This thesis was motivated by the observation that the two uses of marks, as a means of communicating between people and of communicating between people and computers, have never been integrated. This lead to a marking based text editing system, termed MATE for Markup Annotator / Text Editor, which allows users to annotate text documents, edit text documents, and to incorporate annotations as editing commands.

The research involved in designing, building, and testing of this system encompasses three research areas, asynchronous collaborative writing – the intended application, the visibility of markings – the catalyst for the integration of commands and annotations, and common interaction languages which can be understood by both people and computes – a research topic of which this thesis is an example. In addition to concentrating on one particular area of research, we are also interested in the interactions among them.

MATE is intended to support the following asynchronous collaborative writing scenario:

- The principal author creates a draft of the document, and then sends copies to the other collaborators.
- These collaborators review the document, annotate it, and then return the annotated document to the principal author.
- The principal author then incorporates the suggested changes into the document.

The general design of MATE supports each of the above three functions; it allows users to edit text documents, to annotate electronic text documents, and to incorporate the annotations into the new version of the document. It does this by providing three modes of operation, one for each function.

# Chapter 2

# Markings

In Chapter 1, we stated our intention to integrate the two uses of markings – as annotations, and for specifying commands – in a text editing system for collaborative writing. In this chapter, our reasons for using marks are given, along with comparisons with other methods such as Graphical User Interfaces, and Speech Based Interfaces. Next we examine how markings are currently used in making annotations and for specifying commands. Finally, the theory underlying the integration of annotations and commands is developed.

## 2.1 Why Markings

There are several reasons for focusing our research on markings; they are already in common use in annotating documents, and have several properties which support their use as annotations, for specifying commands, and for integrating these two uses. But mouse and keyboard based interfaces and speech based interfaces could also be used, and are therefore compared with our marking based approach.

### 2.1.1 The Everyday Use of Markings

People use marks everyday to make comments, notes, etc. This frequent usage provides the following benefits:

1. people can make marks (A)
2. people understand marks (B)

An additional effect from this everyday use is that a person (A) expects another person (B) to understand A's marks. This expectation may actually lead to mistakes, or misinterpretations by the reader of the annotations. The annotator expects the reader to interpret the annotation properly, but has no easy way of knowing if the reader actually interprets it correctly.

There are also benefits from using a set of markings for commands which come from the markings in everyday use:

- The skills obtained from this everyday usage are learned over a long period of time. Therefore it is beneficial to base our use of marks on these everyday skills so that users require minimal training.
- By using a set of marks which can be understood by both people and computers, a common language is achieved. Such a language would reduce or eliminate the need to convert information intended for a person into commands to specify to a computer.

### 2.1.2 Properties

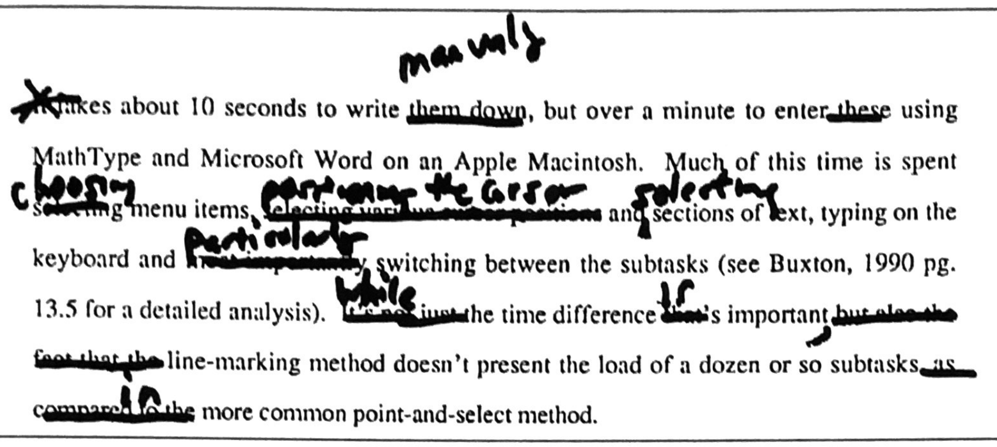
Markings have many characteristics which make them well-suited as annotations and for specifying commands.

Markings are useful as annotations because:

- *Markings are Easily Made and Understood by Most People:* This was noted above in Section 2.1.1 on the Everyday Use of Markings.
- *Markings are Expressive:* Markings can convey much information such as position, size, attributes such as boldness, italics, etc.
- *Markings are Qualitatively Distinct from Typeset Text:* Markup annotations are visually distinct from typeset text. This allows one to view a marked up document and easily distinguish the markings from the text. In fact, the markings tend to stand out in the foreground as the typewritten text fades to the background. This characteristic is termed as a *figure-ground distinction*, and can be noticed in typewritten documents annotated with a pen as shown in Figure 2.1.
- *Markings Provide an Overall View:* Someone viewing a marked up document can obtain an overall picture of where the annotations are, how many changes are being suggested, the overall level of detail of the annotations, etc.

Markings are useful for specifying commands to a computer because:

- *Markings are Expressive:* Expressiveness indicates how much information can be obtained from a method of input. Marks can specify many parameters in a wide variety of ways. For example, if an application uses marks to enter characters and



manualy

~~It t~~akes about 10 seconds to write ~~them down~~, but over a minute to enter ~~these~~ using MathType and Microsoft Word on an Apple Macintosh. Much of this time is spent choosing ~~selecting~~ menu items, positioning the cursor ~~selecting various cursor positions~~ and selecting sections of text, typing on the keyboard and particularly ~~most importantly~~, switching between the subtasks (see Buxton, 1990 pg. 13.5 for a detailed analysis). While ~~It's not just~~ the time difference Is ~~that~~'s important, ~~but also the fact that the~~ line-marking method doesn't present the load of a dozen or so subtasks ~~as compared to the~~ in more common point-and-select method.

**Figure 2.1:** The Qualitative Distinction Between Marks and Typewritten Text
*In a marked up document the marks tend to stand out from the typewritten text. This characteristic is termed a figure-ground distinction.*

symbols, then the position and size of these characters is given by the position and size of the mark. To show the difference between this approach and that of a typical direct-manipulation interface, 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, 1991) for a detailed analysis). Time is only one aspect; the cognitive load of learning, planning, and executing these subtasks is the major drawback of the point-and-select method. Marking does not require this cognitive load as the task is simpler.

Often, a single mark is enough to specify all of the parameters of an action. An example of this is shown in Figure 2.2. In this example, the action, source and destination are all specified by a single mark.

Marks can have an orientation, thickness, size, shape, position, and other attributes. In addition each mark can have several segments with their own attributes as is the case with cursive script. Also marks may form different meanings in association with other marks. For example a straight line may mean underline a piece of text as shown in Figure 2.3(a), or it may mean to delete a piece of text as shown in Figure 2.3(b).

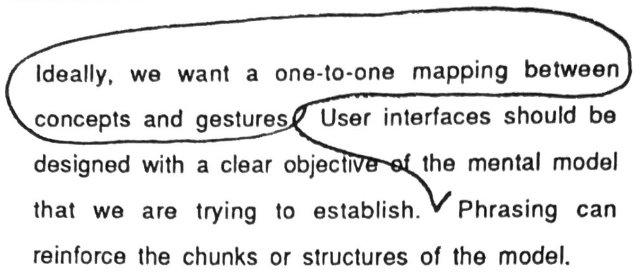
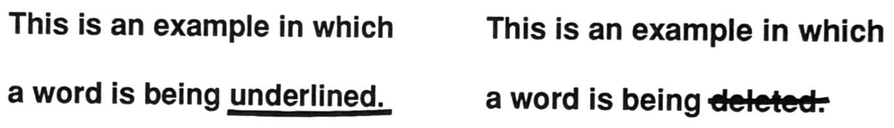

Ideally, we want a one-to-one mapping between concepts and gestures. User interfaces should be designed with a clear objective of the mental model that we are trying to establish. Phrasing can reinforce the chunks or structures of the model.

**Figure 2.2:** Chunking and Phrasing in a Move Command
*A marking allows several components to be "chunked" into one action. The proofreader's move marking captures three components normally articulated by separate commands: the selection (the text to be moved) the command (move), and the destination (where the text is moved) (Buxton, 1991).*


This is an example in which a word is being underlined.

This is an example in which a word is being ~~deleted.~~

(a) (b)

**Figure 2.3:** Markings in Context
*The meaning of the horizontal line depends upon its placement. If it is underneath a line of text it is interpreted as an underline command, if it is on the text it is interpreted as a delete command.*

- *Markings are Direct:* Markings are specified directly on the object(s) of interest (Wolf and Morrel-Samuels, 1987). For example, the cross-out mark in Figure 2.4 is made directly on the word to be deleted. In contrast, with many forms of "direct-manipulation" the objects and commands are specified separately by first selecting the object and then the command.

- *Markings have an Inherent Syntax:* The syntax of commands is determined by the spatial form of the markings. This is illustrated with an example of the move command.

In a command-line system there are many alternative formats of the move command. Two of the most common are:

Move (from) Source (to) Destination.
Move (to) Destination (from) Source.

It is not clear which format a particular command-line system expects.

However, with a markup method, one naturally specifies the source first and the

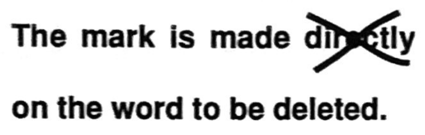


**Figure 2.4:** The Directness of Markings
*The command is made directly on the text to be deleted.*

destination second, and the move command is specified at the same time as the parameters, as shown in Figure 2.2. This "natural order of interaction" (Buxton, 1990) is more in line with thinking of the entire command as one process. One performs actions on or to objects. One way to think of this is that dialogues such as command-line, point-and-select, and verbal are generally serial, specifying one thing at a time, whereas a dialogue based on markings is more parallel as many actions/objects/parameters can be specified at the same time. With a parallel dialogue, the syntax becomes simpler.

### *The Visibility of Markings*

The fact that markings are visible is the key to integrating annotations with commands. Markings must remain visible in order to be useful as annotations. This is not necessary when markings are used as commands – the markings can be removed once the command is performed. The two uses can be connected by observing that an annotation is a command which has not yet been processed. Annotations can therefore be thought of as deferred commands. In terms of the different modes of MATE, Annotation Mode can also be termed *Deferred Mode*, and Edit Mode can also be termed *Immediate Mode*.

The integration can be carried one step further by keeping the processed markings visible. This completes the connection as both processed and unprocessed markings can be treated as annotations and commands. As will be shown in later chapters, this allows many new features such as *Undo by Selection*, and a *Graphical History Mechanism.*

Note that our use of markings is constrained by our reliance on the visibility property. The main constraint is that the markings must be visually distinct from each other. This is already necessary for markings used as annotations, but this restricts the markings available as commands. Marking commands cannot use timing or directional information for interpreting the markings.

### 2.1.3 Comparison With Other Methods

As noted earlier, markings are currently the most common method of annotating documents. This in addition to the above-mentioned properties makes markings a good method to use. But there are other viable methods which should be investigated, namely mouse and keyboard based interfaces , and speech based interfaces.

*Mouse and Keyboard Based Graphical User Interfaces*

The typical Graphical User Interface (GUI) does not have the visibility property of a marking based interface, as button clicks and menu selections are not visually persistent. But this can be circumvented by providing commands to make annotations. One system which does this is the *Collaborative Annotator* (Kosarek, et al., 1990). This is a multi-user document review system which can be used in collaborative writing. First the document is scanned into the system. Then the users can annotate the document using the *Collaborative Annotator.* The *Collaborative Annotator* is the mouse and keyboard equivalent to Wang *FreeStyle* in that they both use scanned in bit mapped images of documents and only support annotations.

The main difference between mouse and keyboard based interfaces and marking based interfaces is that with a mouse and keyboard based interface a user must use commands to make annotations, whereas in a marking based interface the annotation is simply whatever markings the user makes. For example, a delete annotation can be visually represented as a horizontal line in both types of interfaces, as shown in Figure 2.3(b). In a marking based interface, the user just draws the horizontal line, whereas in a typical graphical user interface, the user would select the text and then apply the delete annotation to that text usually via a menu selection. The mouse and keyboard method may seem acceptable in a single example, but this method requires that all types of annotations be predefined and accessible as commands. It is this fact, that markings do not require commands, which provides the flexibility to make any type of annotation.

*Speech-Based Interfaces*

Speech is a very good means of communication among people, therefore it makes sense to use speech as a method of annotating a document. In fact, speech is actually good for the types of annotations for which mark-up annotations are poor, namely general comments and "wordy" or long comments. But speech is not visible and is poor in specifying locations, and would therefore need to be used in conjunction with a mark-based or mouse based interface.

Using speech for commands has two main problems:

- currently only discrete speech recognition has progressed far enough to be useful,
- and speech is poor at specifying locations.

Therefore a more ideal system would use a combination mark-based and speech-based interface. However, as the speech-based-interface is dependent upon the mark-based interface, we decided to first see what could be gained with a purely mark-based approach. A hybrid system is left as future work.

## 2.2 Current Usage of Markings and Previous Work

Integrating annotations with commands has not been studied before, except for related work by Goodisman (Goldberg and Goodisman, 1991; Goodisman, 1991). Therefore we examine the current usage and previous work done with markings used as annotations and with markings used to specify commands to applications. Finally we examine the issues brought forth by Goodisman's work.

### 2.2.1 Making Annotations

As there are only a few computer applications which support markup annotations, we examined marked up paper documents to gain further insights. Two useful observations to note are that annotations can range from very specific to very general, and that the available space to make the markings is an issue.

*Levels of Detail*

Annotations can range from very general comments applying to the entire document, as shown in Figure 2.5, to very specific corrections such as removing the quotes around "attached", Figure 2.6. There are several factors that characterize the various levels of detail. These are:

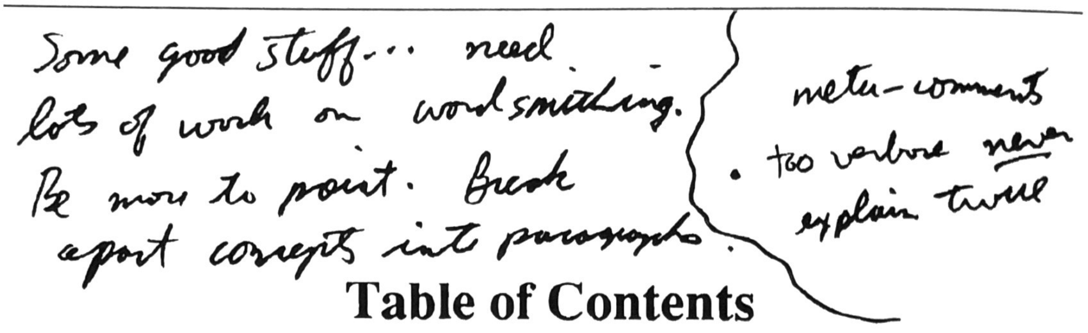
- *Position Attachment:* Annotations apply to a specific section of the document, ranging from the entire document – Figure 2.5, to a paragraph – Figure 2.7, to a single character – Figure 2.6.
- *Length of Comment:* Some comments are brief – "Mush" in Figure 2.7 and "what 3?" in Figure 2.8. Others are very long and contain much information – Figure 2.5 and "there is also ...." in Figure 2.8.
- *Comment vs. Action:* Some annotations are to be treated as a discussion between the

annotator and the reader – Figures 2.5, 2.7, and 2.8. Whereas some annotations are to be treated as specific editing commands – Figure 2.6.

Each of the various methods for annotating documents tend to be stronger at supporting certain levels of detail over others.

Speech is very good for more general, lengthy comments applying to sections of text of paragraph size or larger. Speech is also good for replacing text, but is very cumbersome for specifying details such as punctuation. Speech also requires support to attach the annotation to a particular section of the document.

Typed annotations are reasonable for general comments. They generally take more time



Some good stuff... need lots of work on wordsmithing. Be more to point. Break apart concepts int paragraphs.

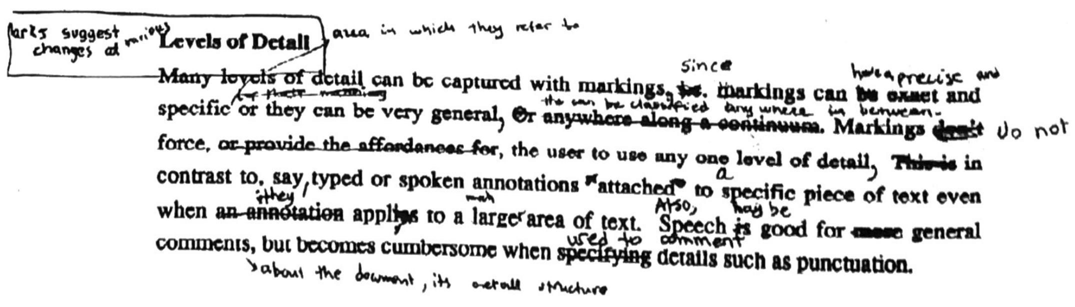
meta-comments

• too verbose never explain twice

Table of Contents

1 Introduction

**Figure 2.5:** General Comments Applying to the Entire Document
*These comments do not apply to any one section of the document, but rather to the whole document.*

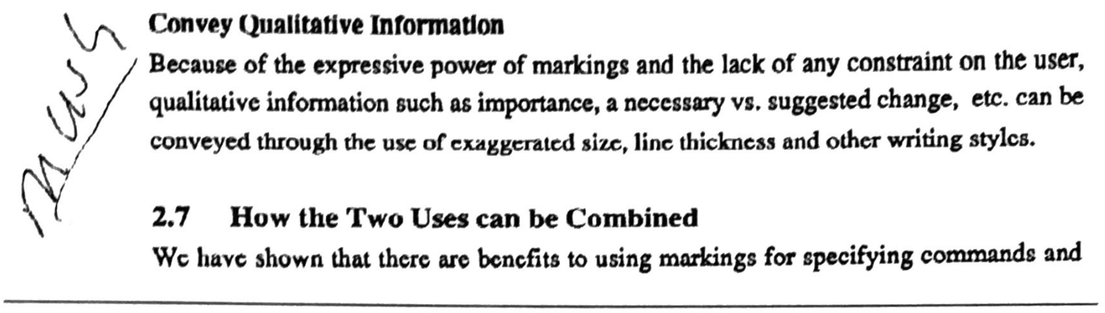


Marks suggest changes at various area in which they refer to

**Levels of Detail**

~~Many levels of~~ detail for their meaning can be captured with markings~~, be~~. Since markings can ~~be exact~~ have a precise and specific or they can be very general, ~~Or~~ the can be classified ~~anywhere along a continuum~~ any where in between. Markings ~~don't~~ Do not force, ~~or provide the affordances for~~, the user to use any one level of detail, ~~This is~~ in contrast to, say, typed or spoken annotations ~~"~~attached~~"~~ to a specific piece of text even when ~~an annotation~~ they apply to a larger much area of text. Also, Speech ~~is~~ may be good for ~~more~~ general comments about the document, its overall structure, but becomes cumbersome when ~~specifying~~ used to comment details such as punctuation.

**Figure 2.6:** Detailed Corrections to Text
*These annotations are detailed corrections to the text. Annotations at this level usually apply to words, such as "attached", and characters "applies / apply, large / larger".*



Mush

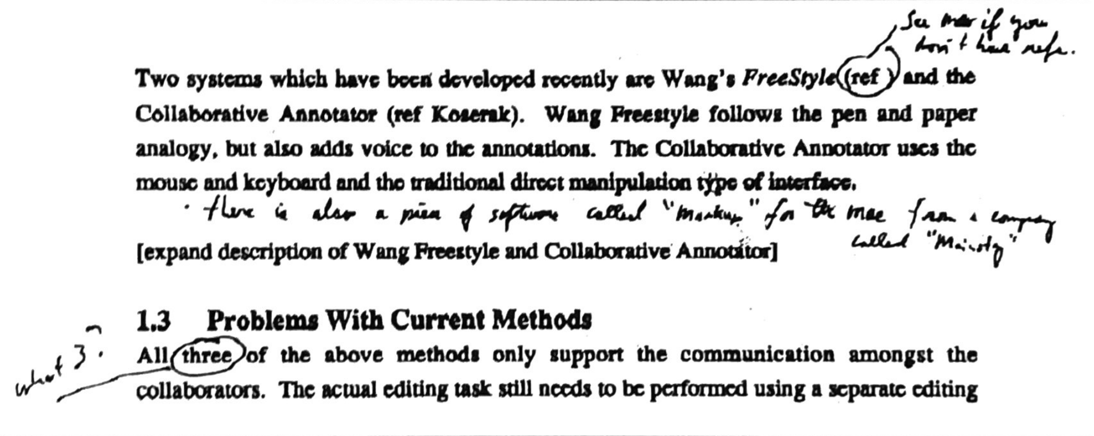
**Convey Qualitative Information**

Because of the expressive power of markings and the lack of any constraint on the user, qualitative information such as importance, a necessary vs. suggested change, etc. can be conveyed through the use of exaggerated size, line thickness and other writing styles.

**2.7 How the Two Uses can be Combined**

We have shown that there are benefits to using markings for specifying commands and

**Figure 2.7:** General Comment Applying to a Particular Section of Text
*The comment "Mush" applies to a single paragraph.*



See me if you don't have refs.

Two systems which have been developed recently are Wang's *FreeStyle* (ref ) and the Collaborative Annotator (ref Koserak). Wang Freestyle follows the pen and paper analogy, but also adds voice to the annotations. The Collaborative Annotator uses the mouse and keyboard and the traditional direct manipulation type of interface.

· there is also a piece of software called "Markup" for the mac from a company called "Mainstay"

[expand description of Wang Freestyle and Collaborative Annotator]

**1.3 Problems With Current Methods**

what 3?

All three of the above methods only support the communication amongst the collaborators. The actual editing task still needs to be performed using a separate editing

**Figure 2.8:** Specific Comments
*These comments apply to specific items within the document. They can range in length from brief, comments such as "what 3?", to long comments, such as "there is also a ...".*

and effort than speech, but they do have some advantages over spoken annotations. Typed annotations are very good for replacing text including punctuation and spelling corrections. Also typed annotations are visible, which allows the reader to have an overall view of the annotations.

For general, lengthy comments, markings take more time and effort than both typed annotations and spoken annotations. Markings are more well-suited for annotations which are more detailed, and especially for annotations specifying actions. But the important fact is that markings can support all levels of detail reasonably, whereas spoken and typed annotations are poor for certain levels of detail.

These observations support the conclusion that a system combining the various forms of annotations is the best approach. In fact, both Wang FreeStyle and the Collaborative

Annotator support several forms of annotations. As mentioned earlier, this thesis is only concerned with marking annotations, a system with other forms of annotations is left as future work.

*Markability*

Markability refers to how easy it is to make marks in the space available between the lines of a text document. One way of thinking about markability is that visible information, such as the text of a document and the changes to or comments about this document, requires space. On paper, markability is facilitated by the use of double and triple spacing in draft versions of the document. But what if one wants to make marks on a formatted version of the document? There is not much that can be done in this case. However, solutions for electronic documents do exist. For example, a text editor could have a draft mode which places more space between the lines to allow marks. Markability may force the working version of a document to be different than the presented version of a document. This is an issue which needs to be considered for all systems which use markings.

### 2.2.2 Specifying Commands

Markings are used to specify commands for a wide variety of applications, such as Text Editing (Welbourn and Whitrow, 1988; Carr, 1991; Wolf and Morrel-Samuels, 1987), Musical Notation (Buxton, et al., 1979; Wolf, et al., 1989), Mathematical Notation (Wolf, et al., 1989), and Educational Applications (Chow and Kim, 1989). There are many issues which need to be addressed when using markings for commands; the two that affect this thesis the most are the Dependence on Recognition Technologies, and the Management of the Interaction Dialogue.

*Dependence On Recognition Technologies*

Many marking-based applications require state of the art technologies in order for them to be realized and thus much research is being conducted in the areas of character recognition (Tappert, et al., 1990; Ward and Blesser, 1985), and marking recognition (Rubine, 1990; Kim, 1988). Even as the recognition technologies improve it is important to realize that the recognizers will make mistakes, and that the interface must be able to allow users to correct errors.

*Character Recognition:* Recognition rates for the recognition of boxed discrete characters has reached over 95% (Tappert, et al., 1990). The recognition rates for nonboxed characters and cursive script is much lower and usually must be adapted for each user. These recognition rates place restrictions on the way that recognizable characters can be written. Also, the fact that the recognition rates are not close to 100% accuracy means that error recovery and feedback become very important. Deferring the recognition of the character amplifies this issue of feedback and error recovery, because the user who writes the character does not receive any feedback. Also, the user viewing the marks has no way of knowing if the recognizer is conveying the annotator's intentions. This is an important issue and is dealt with in more detail in the Design and Implementation sections.

*Marking Recognition:* Marking recognition refers to the recognition of all markings which are not characters. As opposed to characters, markings may vary in size, shape, and orientation, which make recognition more difficult. Another problem with mark-recognition is that it is usually not sufficient to simply recognize a mark; characteristics of the mark also need to be extracted. For example, the move mark shown in Figure 2.2 specifies the source and destination in addition to the action. All of these parameters need to be extracted from the mark.

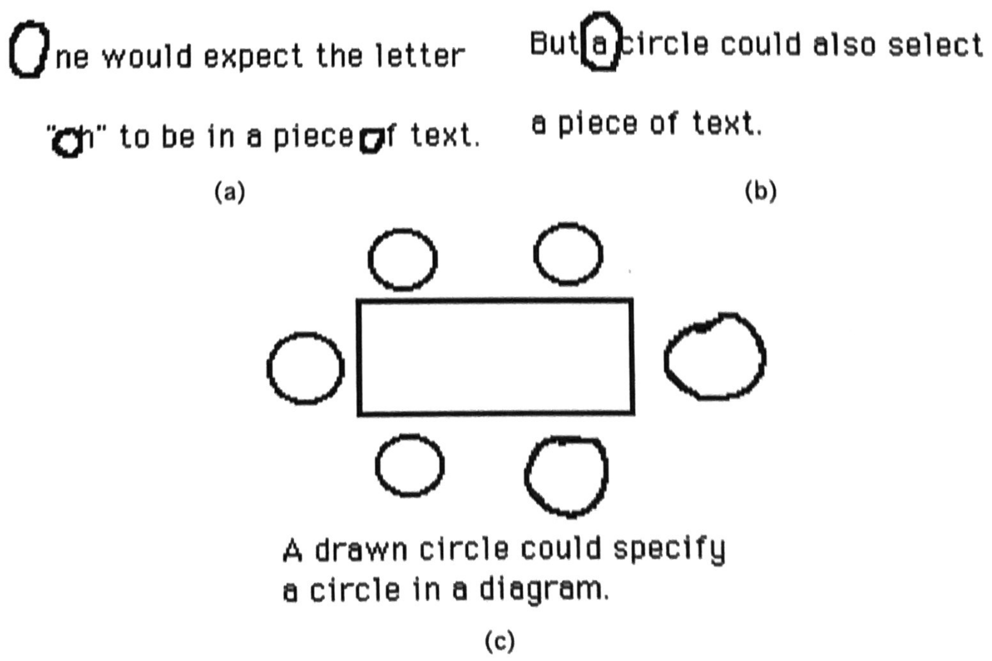

One would expect the letter
"oh" to be in a piece of text.
(a)
But a circle could also select
a piece of text.
(b)
A drawn circle could specify
a circle in a diagram.
(c)

**Figure 2.9:** Different Interpretations of the Circle Mark
*Depending upon the context, a circle mark could be specifying (a) the letter "O', (b) a selection of text, or (c) a circle.*

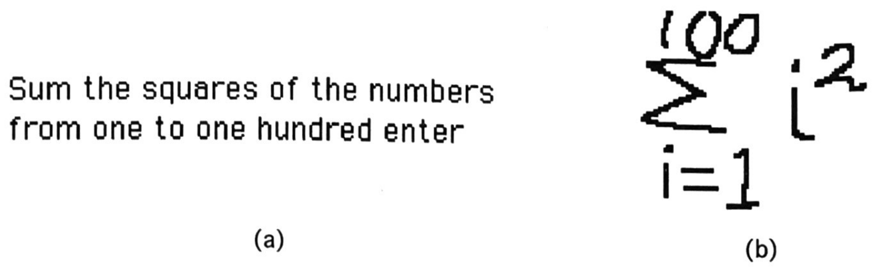
Most mark recognition techniques are different than character recognition techniques. This causes problems when both characters and marks are allowed, because a mark might be recognized as both a character and as a mark. For example, the circle in Figure 2.9 could be specifying a scope, a circle, or the letter "O". If both a character and a mark recognizer are used, a means of comparison is necessary to choose the final interpretation.

*Dialogue Management*

Most dialogues with a computer are sequential in nature. In contrast, interactions using markings are more spatial. To better understand this distinction, consider entering a command to sum the squares of the numbers from 1 to 100. In a command-line or verbal interface, the command would look something like what is shown in Figure 2.10(a). Note that the command would be typed or spoken in the order shown and followed by an event to perform the execution. In contrast, the markup command would look like what is shown in Figure 2.10(b). What is important to note is that any of the parameters could be entered in virtually any order as long as they are placed in the correct position.

The fact that interactions with markings are more spatial than temporal, brings about many issues which are not normally present in temporal dialogues. Rhyne (1987) points out syntactic compression, contextual effects, closure, and embedded dialogues as some of these issues.

- *Syntactic Compression and Segmentation:* Syntactic compression refers to the ability to specify several parts or all of a command with a single mark. These marks need to be separated into their constituent parts, which can be a complex process. Figure 2.2 is an example of compressing a move command into a single mark. Syntactic segmentation refers to the fact that several marks may make up a single character or


Sum the squares of the numbers
from one to one hundred enter

$$\sum_{i=1}^{100} i^2$$

**Figure 2.10:** A Comparison of Verbal vs. Markup Dialogue
*The verbal dialogue (a) follows a sequential order, whereas the markup dialogue (b) is spatially ordered, the sequence in which the terms are written is irrelevant.*

part of a marking. This occurs when one "crosses the t's" and "dots the i's and j's" after writing an entire word or sentence.

- *Contextual Effects:* For a spatial dialogue, the spatial context – where the mark is in relation to other visible objects, may be more important than the temporal context – when the mark was made in the current command specification. GO uses spatial context to distinguish between circles, selections, and oh's (Carr, 1991).
- *Closure:* This refers to the method which signals the end of a dialogue. This could be an explicit event such as pressing the enter key on a keyboard, or the lifting of the stylus. It could also be an implicit event such as a certain period of time elapsing. GEdit is an example of a system which uses a combination of implicit - elapsed time, and explicit - pen lifting, closure (Kurtenbach and Buxton, 1991a).
- *Embedded Dialogues:* There are many cases in which the user may temporarily interrupt one dialogue and start another sub-dialogue. A good example of this is in a move command for in which the source and destination are far apart. In this case the user would specify the scope, i.e. source of the command, perform some type of navigation command, and then finish the move command by specifying the destination. Embedded dialogues are difficult for the computer to handle because it must constantly make a decision on whether an event belongs to the current dialogue or is the beginning of a sub-dialogue.

Recognition technologies and dialogue management are not the central issues of this thesis, but must be dealt with in order to address the integration of markings as commands and annotations. These issues are taken into account in our choice of the set of markings used for commands and in the design and implementation of MATE.

### 2.2.2 Integrating Commands and Annotations

Goodisman (Goldberg and Goodisman, 1991; Goodisman, 1991) has developed a stylus-based text editor which allows the execution of markup commands to be deferred. In Goodisman's system the user first selects the editing marks to perform, and then has them all performed at once. Although his emphasis – deferring commands in a text-editor, is different from that of this thesis – integrating annotations and commands, the works are related and he discusses several important issues.

- *The need for feedback:* The user requires some kind of feedback to be able to know how the system interprets the mark.

- *Non-chronological undoing:* Most editing systems only allow the last "n" commands to be undone. As will be shown in later chapters visible marks allow the arbitrary undoing of commands.
- *Disorientation:* Users may lose track of what the document will look like after the editing changes are made. In fact, users tend to manually confirm that the changes have been made, which can be difficult because there are no longer any marks by which to locate these changes.

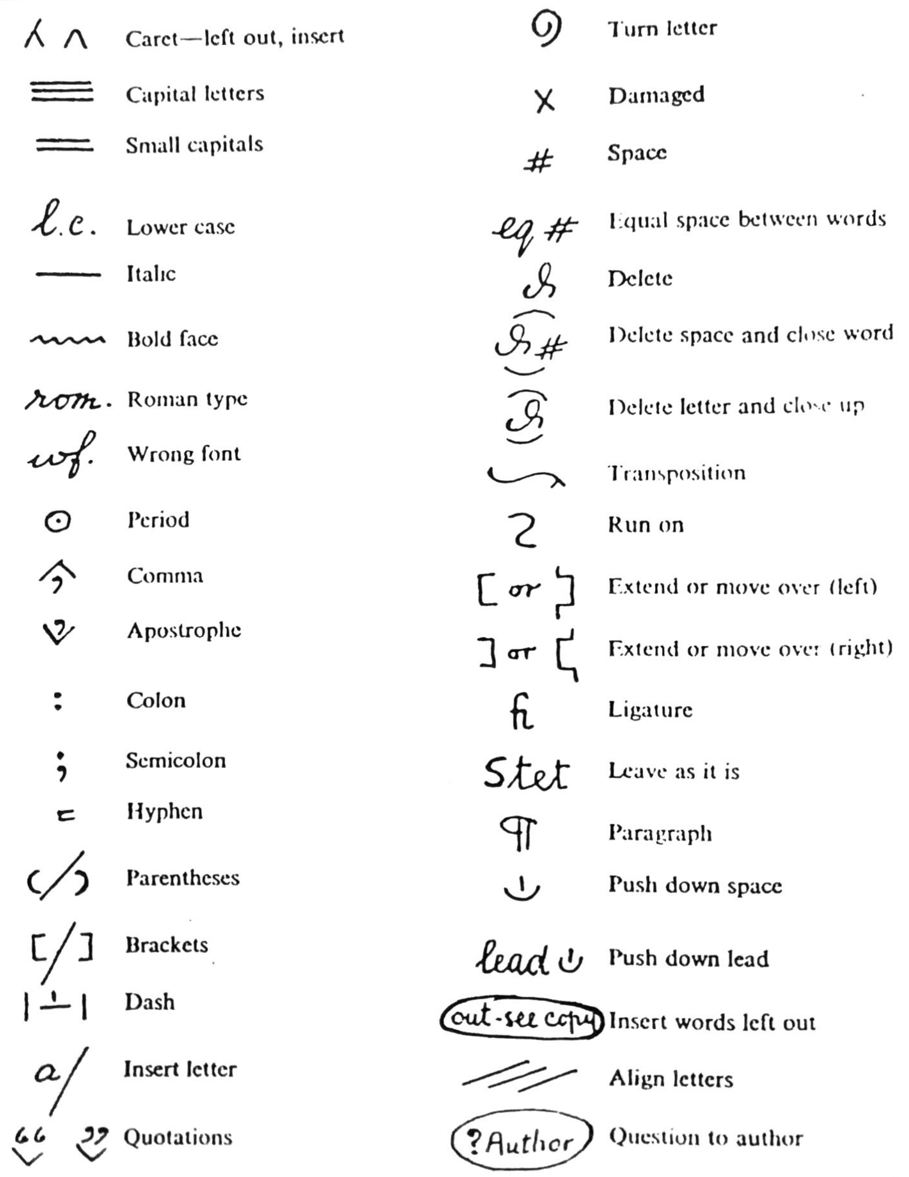


**Figure 2.11:** Symbols Used by Proofreaders in Marking Proof Errors (DuHamel, 1962)

Each of the issues brought forth by Goodisman's work are addressed in this thesis.

## 2.3 Markup Languages

To design the marking set for MATE we examine markup languages for annotations and for commands. Our constraints are the available character and marking recognizers at the time of implementation, the fact that the markings must be visually distinct from each other, and that the markings be understandable to the user.

There are many standard proofreaders' languages, such as the one shown in Figure 2.11 from (DuHamel, 1962). But most people do not learn these standards and use more intuitive, but less functional languages. Wolf (1987) conducted a study to find out what marks were actually used by people for annotating text. The most common of these are shown in Figure 2.12.

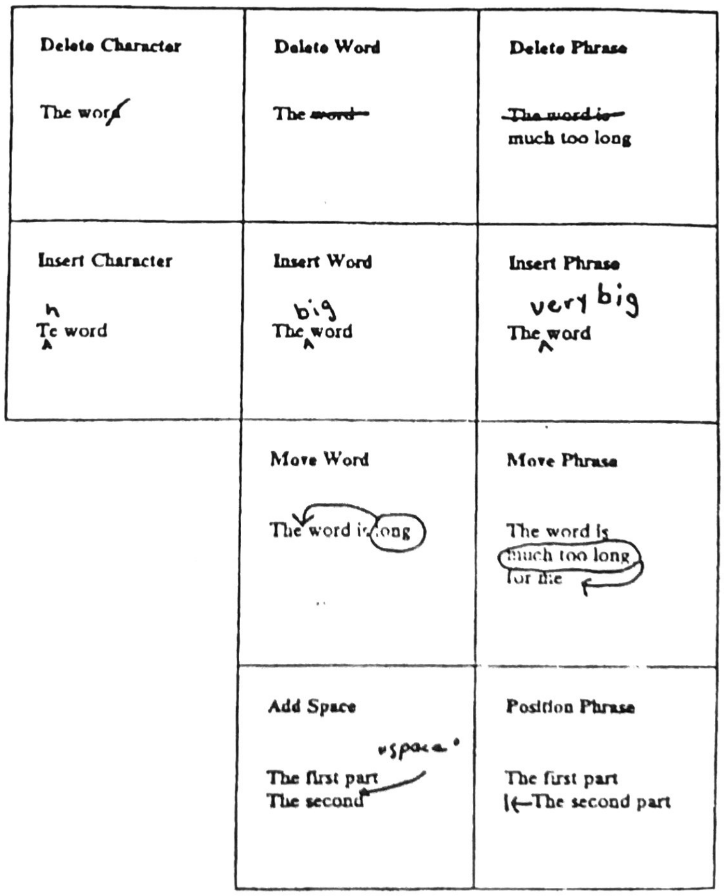
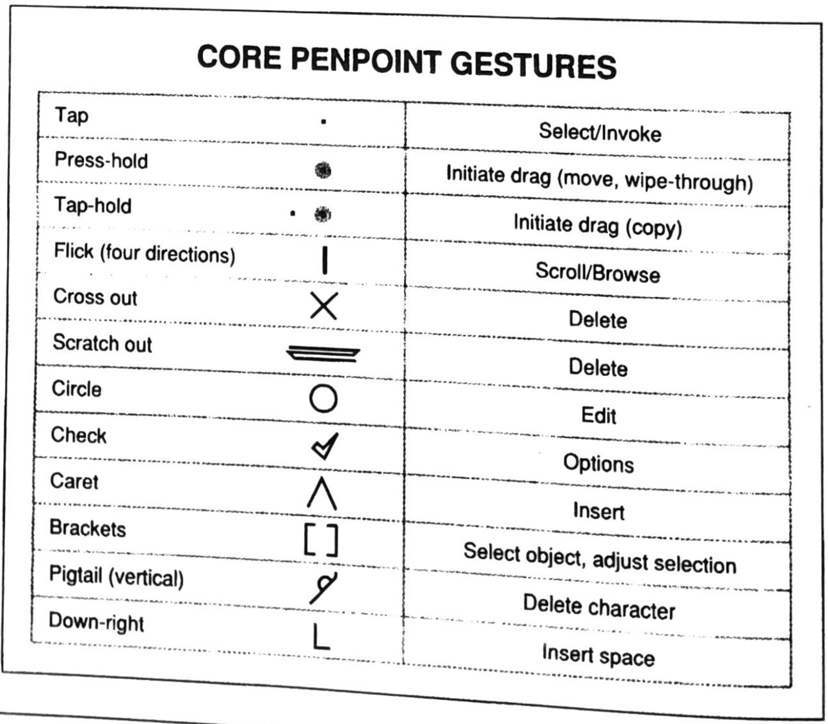


**Figure 2.12:** The Most Common Marks for Ten Editing Operations (Wolf and Morrel-Samuels, 1987)

Marks used as commands to computer applications are different than those used for annotations. Two sets of mark commands are shown in Figures 2.13 (Carr, 1991) and 2.14 (Welbourn and Whitrow, 1988). The main reason for the differences is that people and computers have different strengths and weaknesses in their recognition capabilities. One important point is that many marking and character recognizers use timing and direction information to distinguish between marks. For example, the three PenPoint marks: tap, press-hold, and tap-hold look the same, the only difference is in their timing. Such marks cannot be used as they do not meet the constraint of being visually distinct.

## *The Marking Set Chosen for MATE*

The marking set chosen for MATE consists of the insert caret, a horizontal line for delete, and the move mark , shown in Figure 2.15. The reasons for these three commands are discussed in Chapter 3. Note that each of these marks are common annotation marks (Figure 2.12), and have been implemented in a text editor (Figure 2.14). A character recognizer was not available at the time of implementation, instead characters are typed into a text entry box which appears when the insert caret is made.



**CORE PENPOINT GESTURES**

| | | |
|---|---|---|
| Tap | · | Select/Invoke |
| Press-hold | ● | Initiate drag (move, wipe-through) |
| Tap-hold | · ● | Initiate drag (copy) |
| Flick (four directions) | \| | Scroll/Browse |
| Cross out | × | Delete |
| Scratch out | ≡ | Delete |
| Circle | ○ | Edit |
| Check | ✓ | Options |
| Caret | ∧ | Insert |
| Brackets | [ ] | Select object, adjust selection |
| Pigtail (vertical) | ℘ | Delete character |
| Down-right | L | Insert space |

**Figure 2.13:** Basic Pen Gestures Recognized By PenPoint (Carr, 1991)

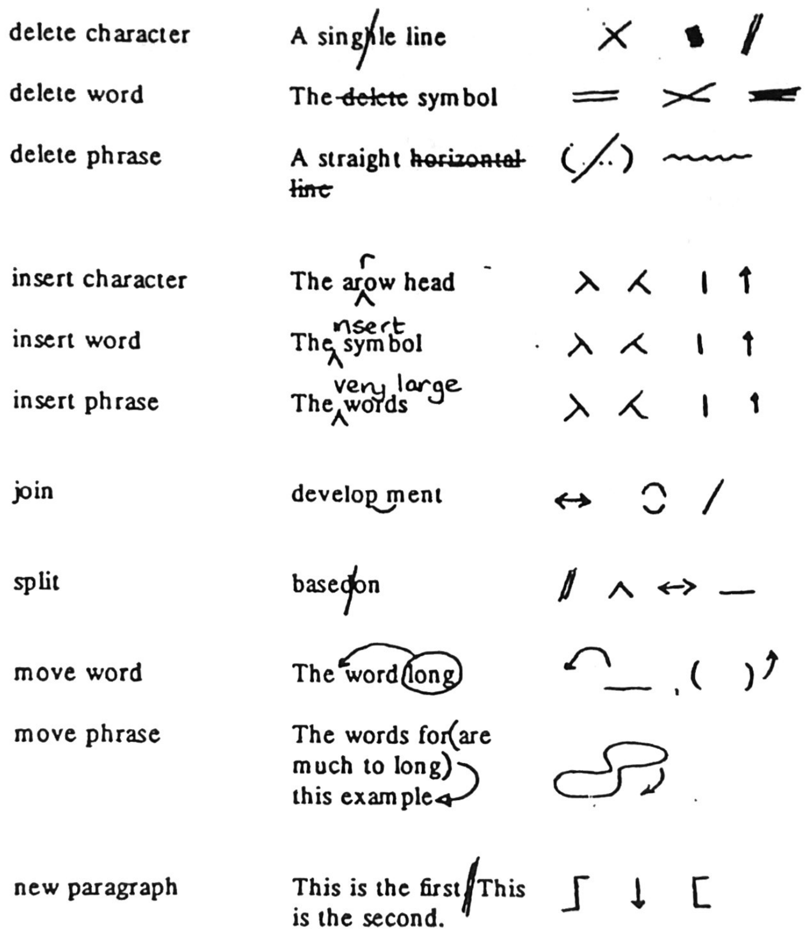


**Figure 2.14:** Editing Symbols from a Mark Based Text Editor (Welbourn and Whitrow, 1988)

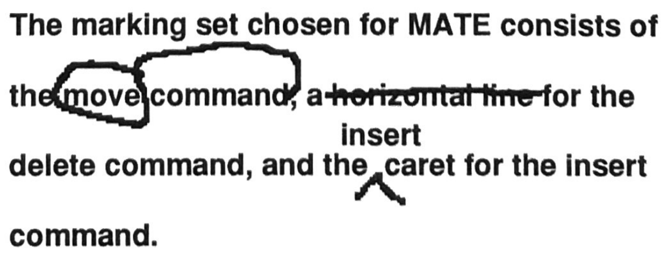

The marking set chosen for MATE consists of the move command, a ~~horizontal line~~ for the delete command, and the $\overset{\text{insert}}{\wedge}$ caret for the insert command.

**Figure 2.15:** MATE's Marking Set
*MATE's marking set consists of the move mark, a horizontal line for delete, and the insert caret. Characters are typed into a text entry box once the insert caret is recognized.*

## 2.4 Integrating the Two Uses

The theory behind the integration is that annotations can be treated as deferred editing commands. This is possible from the fact that marks can be visibly persistent. But there are many practical issues that need to be dealt with in order to achieve the integration:

- *Static vs. Dynamic Documents:* Annotations are made on a static document, whereas editing commands are made on a changing document. This presents many problems, for as a document changes, the annotations may have a new meaning or become obsolete. This is one of the central issues of this thesis and is dealt with in Chapters 3 and 4.
- *Constraints on the Marking Set:* The constraints placed on the marking set are that the markings are visually distinct from each other and that the marking commands do not rely on timing or directional information. These constraints restrict which marks are available, and how the marks are recognized. These restrictions are taken into consideration in Chapter 4 – Implementation.
- *Different Marking Sets Between Annotations and Commands:* As shown in Section 2.3, marks which have traditionally been used for annotations are different than those being used as commands to computer applications. Although some overlap exists, it may not be enough for the integration to be useful. Due to the restricted recognition technology available at the time of implementation, the marking set chosen is small. This issue is examined in Chapter 5 – User Testing, but cannot be fully dealt with until a larger marking set is implemented.

Central to our research is to determine what are the benefits of integrating annotations and commands. From the theory developed in this chapter, we can already expect certain benefits:

- *Enhance the Annotation Reader's Understanding:* The annotation's creator expects the reader to understand the markings, but the reader could possibly misinterpret or fail to understand the writer's intentions. The computer's ability to understand the edit marks can lessen the possibility of a misunderstanding between the annotation's writer and reader. Table 2.1 shows how this is accomplished. Also the writer of the annotations can verify the computer's interpretation of a mark, thereby reducing the chance of the computer misinterpreting a mark.
- *Enhance the Annotation Writer's Understanding:* The exact results of annotations may not be clear to the annotations' creator. With a mechanism to preview the results

| **Understanding** | **Reader** | | |
|---|---|---|---|
| **Computer** | **Understands** | **MisInterprets** | **Does not Understand** |
| **Understands** | mutual understanding | Computer understands, may help the reader understand the marking | |
| **MisInterprets** | user will ignore computer's misinterpretation | user and computer may have different misinterpretations, which may cause the user to possibly try a different interpretation | User may accept the computer's misinterpretation |
| **Does not Understand** | computer neither aids nor hinders user interpretation | | |

**Table 2.1:** Possible Interpretations By the Reader and the System
*The reader of an annotation is able to verify their interpretation by examining the computer's interpretation of that annotation. This can improve the reader's understanding in some cases, and thus helps reduce the possibility of misinterpreting the annotation.*

of these annotations, the annotator can gain a better understanding of the annotations' consequences. This mechanism also encourages the annotator to use marks which the computer will understand to gain its benefits.

- *Reduction of Effort:* With the integration, the need to convert the annotations into editing commands is reduced, which in turn reduces the effort required in editing the document.
- *"What if?" Scenarios:* Having deferred edits which can be selected and unselected allows one to try various changes and return to the original state of the document. This can be particularly useful when a user is incorporating annotations from several sources and wishes to compare the results of each.

## 2.5 Summary

Markings have several properties which make them well suited for annotations and for specifying commands. In particular, the fact that marks are visible provides a way to integrate annotations and commands. An annotation can be considered as a deferred command. Therefore MATE's Annotation Mode can also be called Deferred Mode, and Edit Mode can also be called Immediate Mode.

A comparison with mouse and keyboard interfaces and speech based interfaces revealed that a hybrid marking and speech based interface would be the best. However, we decided to concentrate our efforts on just a marking based interface first, leaving a hybrid system for future work.

MATE's marking set is shown in Figure 2.15. It consists of the insert caret, horizontal line for delete, and the move mark. As a character recognizer was not available at the time of implementation, a keyboard is used to type characters into a text entry box.

There are several practical issues concerning the integration:

- *Static vs. Dynamic Documents:* As a document is edited, the annotations made on a specific version may have new meanings or become obsolete.
- *Constraints on the Marking Set:* The markings in MATE's marking set must be visually distinct from each other and the recognition techniques cannot rely on timing or directional information.
- *Different Marking Sets Between Annotations and Commands:* Integrating the two uses may not be practical if there is not enough similarity or if a common marking set cannot be developed between those used for annotations and those for specifying commands.

Several benefits can be expected according to the theory developed in this chapter:

- *Enhance the Annotation Reader's Understanding:* As shown in Table 2.1, the computer's ability to understand the edit marks can lessen the possibility of a misunderstanding between the annotation's writer and reader.

  *Enhance the Annotation Writer's Understanding:* With a mechanism to preview the results of one's annotations, the annotator can gain a better understanding of the annotations' consequences.
- *Reduction of Effort:* With the integration, the need to convert the annotations into editing commands is reduced, which in turn reduces the effort required in editing the document.
- *"What if?" Scenarios:* Having deferred edits which can be selected and unselected allows one to try various changes and return to the original state of the document.

# Chapter 3

# Design[†]

The general design of MATE was described in the preceding chapters. To summarize:

- MATE is intended to support the collaborative writing scenario in which one person is in control of the document.
- By designing and implementing MATE, we intend to determine how annotations and commands can be integrated, and to identify the benefits and constraints of such a system.
- MATE supports the collaborative writing scenario through three different modes of operation:
  - Edit Mode (Immediate Mode) supports the editing of text documents
  - Annotation Mode (Deferred Mode) supports the annotating of text document
  - Incorporation Mode supports the incorporation of annotations into a text document

In this chapter the reasons for these design decisions and a more detailed description of the design are given. First an informal case study of the collaborative writing scenario is described and studied. Next, the requirements that MATE should satisfy are explained. Finally, the designs of the overall system, Annotation Mode, Edit Mode, and Incorporation Mode are given.

## 3.1 Informal Case Study

In order to gain more insights on how the system should be designed and on issues that might occur in the collaborative writing scenario, an informal case study was done. Paper copies of a version of a conference article were given by the principal author to five

† An earlier description of the design of MATE is given in (Hardock, 1991).

collaborators. Each collaborator then annotated the document by writing with a distinctly coloured marker on transparencies overlaid on each page of the paper copy. These annotated versions were then given back to the principal author who incorporated the annotations into the document to produce the final version.

There are several points to note about this study:

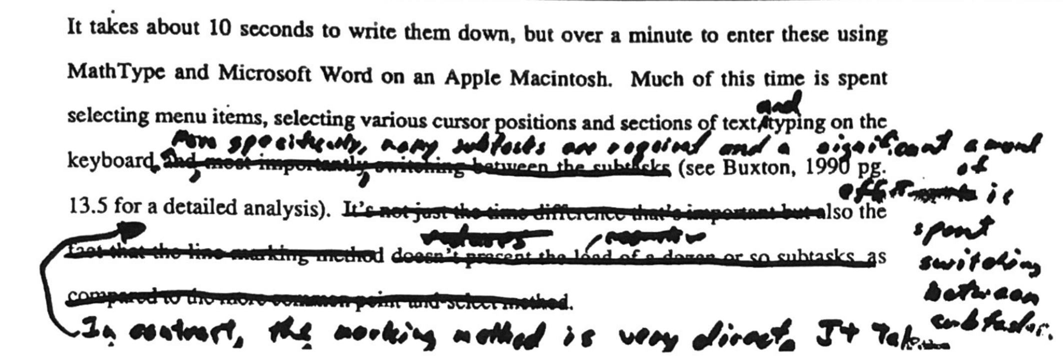
- Except for the use of overlaid transparencies, the study followed what is common in everyday practice.
- By using transparencies, one could place annotations from different sources on top of each other.
- The document was not in its final formatted state but rather in a triple-spaced draft version.

The results of this study were:

- The process of carrying out the study was useful in itself as it helped produce a finished document.
- Viewing annotations from several sources at once is very helpful. However, overlaying annotations from several sources, as opposed to placing them side by side made the annotations difficult to read in some cases as shown in figure 3.1(c).
- Those markings which could be directly translated into editing commands were similar to those found in previous studies, such as Wolf and Morrel-Samuels (1987).

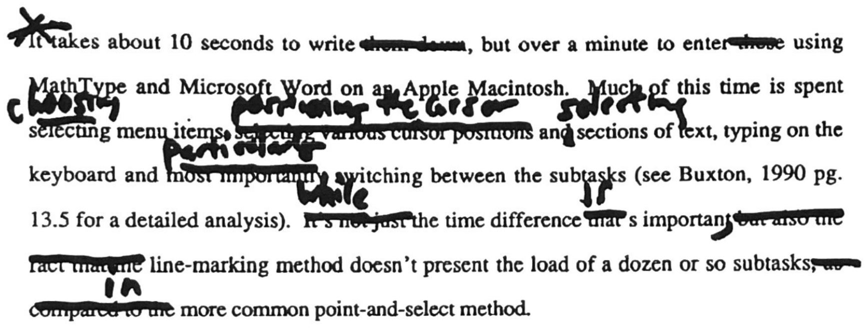
One noteworthy observation is that four of the collaborators knew the technical aspects of the document, whereas the fifth was a professional proofreader not familiar with the content of the document. The technical collaborators gave more general comments about the structure and wording of the document. In contrast, the professional proofreader gave more detailed corrections concerning punctuation and grammar.

The general comments did not translate directly into any specific editing action, rather they required the principal author to rethink what was written and possibly rewrite, delete or add entire sections. The more detailed corrections often translated directly into editing actions by the principal author. It is these types of annotations which are well suited to being interpreted by the system into editing commands. Therefore the usefulness of interpreting mark annotations into editing commands increases as the annotations become more detailed, which tends to occur closer to the end of the writing process.



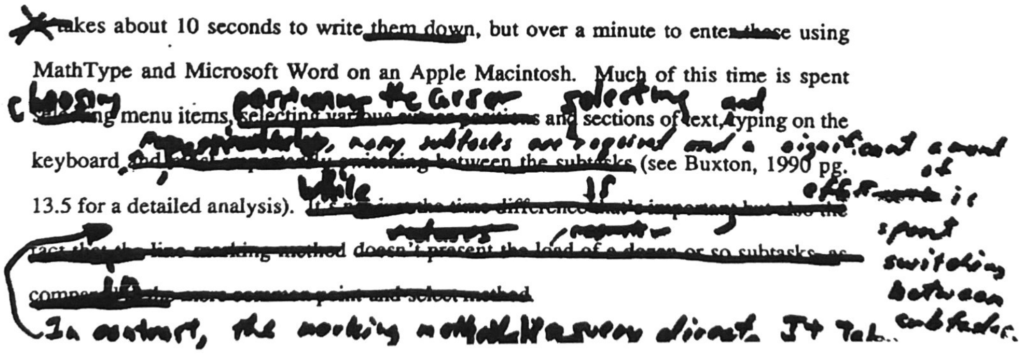
It takes about 10 seconds to write them down, but over a minute to enter these using MathType and Microsoft Word on an Apple Macintosh. Much of this time is spent selecting menu items, selecting various cursor positions and sections of text, and typing on the keyboard. More specifically, many subtasks are required and a significant amount of effort is spent switching between subtasks. ~~and most importantly switching between the subtasks~~ (see Buxton, 1990 pg. 13.5 for a detailed analysis). ~~It's not just the time difference that's important but also the fact that the line-marking method doesn't present the~~ reduces cognitive ~~load of a dozen or so subtasks as compared to the more common point-and-select method.~~
In contrast, the marking method is very direct. It takes...

(a)



~~It~~ takes about 10 seconds to write ~~them down~~, but over a minute to enter ~~these~~ using MathType and Microsoft Word on an Apple Macintosh. Much of this time is spent choosing ~~selecting~~ menu items, ~~selecting various cursor positions~~ positioning the cursor and selecting sections of text, typing on the keyboard and ~~most importantly~~ particularly switching between the subtasks (see Buxton, 1990 pg. 13.5 for a detailed analysis). While ~~It's not just~~ the time difference ~~that~~ is important, ~~but also the fact that the~~ line-marking method doesn't present the load of a dozen or so subtasks, ~~as compared to the~~ in more common point-and-select method.

(b)



~~It~~ takes about 10 seconds to write them down, but over a minute to enter ~~these~~ using MathType and Microsoft Word on an Apple Macintosh. Much of this time is spent choosing ~~selecting~~ menu items, ~~selecting various cursor positions~~ positioning the cursor and selecting sections of text, and typing on the keyboard. More specifically, many subtasks are required and a significant amount of effort is spent switching between subtasks. ~~and most importantly switching between the subtasks~~ (see Buxton, 1990 pg. 13.5 for a detailed analysis). While ~~It's not just the time difference that's important but also the fact that the line-marking method doesn't present the~~ if reduces cognitive ~~load of a dozen or so subtasks as compared to the more common point-and-select method.~~
In contrast, the marking method is very direct. It takes...

(c)

**Figure 3.1:** Combining Multiple Sources of Annotations (Informal Case Study)
*(a) and (b) show annotations from two collaborators. The issue of markability (Section 2.2.1) is apparent by the small hand printing and notes in the margins. Although different coloured markers were used in the study, the situation of overlaying the two sets of annotations is not much better than that shown in (c).*

## 3.2 Requirements

There are four main requirements that MATE should satisfy. These are:

1. To allow users to annotate text with markings.

2. To accept text-editing commands in the form of markings which are visually distinct and consistent with the markings used for a corresponding annotation.

   This requirement implies that the type of markings allowed is restricted. No temporal information is available to the user to distinguish various markings; all the information of the marking must be conveyed visually. This means that information such as tapping of the stylus, and other time dependent input cannot be used as part of an editing command. However, this does not mean that temporal information cannot be used in MATE for other purposes such as navigation and file handling.

3. To achieve a smooth transition from using marks as annotations and as editing commands.

   One of the central ideas of this thesis is that by combining the two uses of markings some effort will be saved. It is not enough to simply have a system that can be used as both an annotation tool and a text-editor. These two uses must be integrated in a way that reduces the effort required to incorporate annotations into a document. This smooth transition will inevitably depend upon the cooperation of the users, as it is up to them to use annotations which can be interpreted as editing commands by the system.

4. To allow users to view annotations in a legible manner from several sources.

   From the informal case study and from what is commonly found in collaborative efforts, the principal author needs to see the annotations from several collaborators juxtaposed with each other. As noted in the case study, legibility becomes more of a problem when dealing with several sets of annotations at the same time. Therefore MATE must provide a way such as selectively showing / hiding annotations to ensure the clarity of the annotations.

## 3.3 Design Alternatives

The asynchronous collaborative writing scenario, and therefore MATE, maps into 3 main modes of operation: annotating, editing, and incorporating annotations. The designs of the Edit and Annotation Modes are relatively straightforward, and are discussed in Section 3.4. However, there are several possible ways to design Incorporation Mode, the main design decision being the number of views or windows to use:

- *Single view:* A single window containing the current edited version of the document and the annotations.
- *Two views:* One window for the original version of the document containing all of the annotations, and the second window for the current edited version of the document.
- *Multiple views:* Similar to the two view approach, but there is a window for each set of annotations.

*Single View*

This alternative has the advantages of requiring the least screen space, and of having the annotations shown directly on the current version of the document. But there are several issues which arise due to the attempt to show the annotations on the current view of the document. As the text of the document changes, so must the annotations. For example, suppose that we have the situation shown in Figure 3.2. If the delete marking is incorporated into the document, what should happen to the move marking? And vice versa? And if the move is executed, what should happen to the comment "which"? If the answer is unclear to the user, we cannot expect the computer to know what to do either.

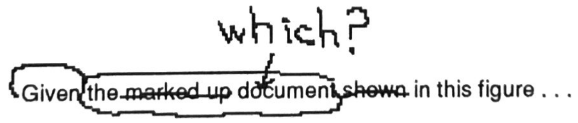

which?
Given the ~~marked up~~ document ~~shown~~ in this figure . . .

**Figure 3.2:** The Difficulty in Updating Annotations
*If the delete marking is incorporated, the move marking will need to be changed, and vice versa. Also the comment "which " should be moved by the move command. There are many more problems, such as what happens to "which" if "document" is deleted, and conflicts among commands such as what happens to the move command if "Given the" is deleted?*

This is a very difficult problem as almost all of the marks may need to be changed or moved. It assumes that enough information can be gathered from the marks to make the correct adjustments, which is usually not the case for marks not intended as editing commands. One way to think of this problem is that annotations are associated with pieces of text. If this text is modified, segmented or removed, the original meaning of the annotation is lost.

*Two Views*

This alternative does not have many of the problems that the single view alternative has, as the annotations remain on a static view of the document. Whereas the single view alternative must interpret every annotation in order to modify it, the two view alternative only needs to interpret those annotations intended as editing commands when they are selected for incorporation.

One observation is that the two view alternative can be considered as a combination of the Annotation and Edit Modes, the main difference being that editing commands can be specified by selecting annotations in the Annotation View. This allows Incorporation Mode to be used for all three functions, annotating, editing, and incorporating.

Although the annotations are not visually modified, their interpretation needs to be converted from the original version of the document to the current version. In some cases, this conversion may not be possible. Such cases are a result of conflicts among the annotations and commands and may cause user confusion and system confusion. Problems with navigating through and relating the two versions will also occur as they become more and more different.

*Multiple Views*

This alternative has similar problems and benefits to the two view alternative. Its two main problems are that it requires more screen space and that the difficulty in finding mappings between the views increases as the number of views increases.

*Discussion*

The three alternatives are not mutually exclusive, in fact the best solution might be to use the single view approach as much as possible, but allow the user to switch to a two view or multiple view approach when needed. But the first step in building such a system is to develop the two view approach first as some of the issues concerning the single view alternative may not have solutions.

The two view alternative's issue of conflicting annotations is handled in Section 3.4.5 and navigation is discussed in Section 4.5. The single view approach and its issues are left for future work.

## 3.4 Design of MATE (Markup Annotator / Text Editor)

Of the three modes in MATE, Incorporation Mode is the only one which has never been designed before and therefore required the generation of design alternatives, discussed in the preceding section. From this design generation, we decided upon the two view alternative for Incorporation Mode.

Annotation Mode and the constraints placed upon it are discussed in Section 3.4.1. Edit Mode is similar to previous marking based text editors, except for the special constraints on its marking set, which is covered in Section 3.4.2. An object oriented approach, discussed in Section 3.4.3, provides the framework for designing MATE as an integrated system. The selection mechanism for incorporation leads to a method for Do / Undo / Redo by selection (Section 3.4.4). Finally the issue of conflicts among commands and annotations is examined in Section 3.4.5.

### 3.4.1 Annotation Mode

In addition to the requirements of most of the previously built annotation tools, Annotation Mode must also support multiple sets of annotations, and store the annotations and text in a form which can later be interpreted as editing commands.

Multiple sets of annotations can be supported by storing each set of annotations separate from the other sets of annotations and from the text. This allows each set of annotations to be retrieved independently of the text and other sets of annotations. Also, by colour-coding each set, the user is able to visually distinguish among the sets of annotations.

To enable an annotation to be interpreted as an editing command, it is stored as the sequence of events which created it. This allows the annotation to be analyzed and interpreted as an editing command at any time.

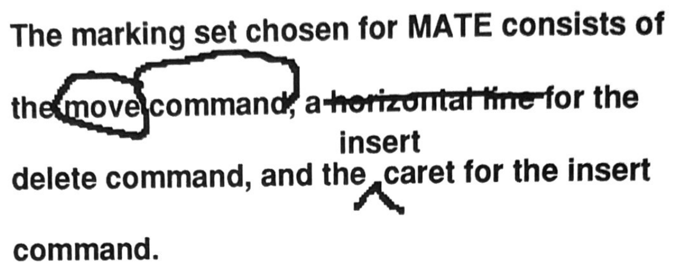
### 3.4.2 Editing Commands

As MATE is intended to be a prototype system, only the three most common annotations were chosen as MATE's set of commands, *delete, move, and insert*. The marks used for these commands are shown in Figure 3.3. These marks were chosen based upon what is found in common usage, and the following three constraints:

- All editing marks must be visually distinct from each other.
- The marks used for editing commands should be the same as those used for

annotations.

- In Incorporation Mode, the Edit View must be able to interpret markings sent to it by the Annotation View.



The marking set chosen for MATE consists of

the move command, a ~~horizontal line~~ for the

insert

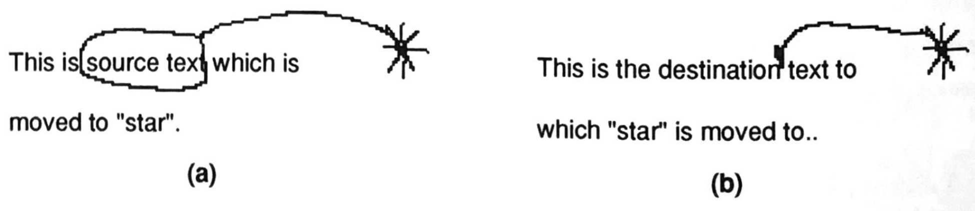
delete command, and the caret for the insert

command.

**Figure 3.3:** MATE's Marking Set
*MATE's marking set consists of the move mark, a horizontal line for delete, and the insert caret. Characters are typed into a text entry box once the insert caret is recognized (From Figure 2.15).*

*Broken Move and Placeholders*

In some cases the source and destination of a move are on different pages[†]. There are several ways of dealing with this in a text editor, such as cutting and pasting, or dragging the source text through several pages to the destination. In an annotated document, the common method is to specify a move from the source to a *placeholder symbol* such as a star, as shown in figure 3.4(a), and then a move from this symbol to the destination as shown in figure 3.4(b).



This is source text which is

moved to "star".

(a)

This is the destination text to

which "star" is moved to..

(b)

**Figure 3.4:** Moving text across pages using *Move to Star* and *Move from star.* *When annotating a document, the move annotation is sometimes broken into two parts. In (a), the source text is "moved" to a placeholder symbol such as a star. In (b), the move command is continued by specifying the placeholder symbol and the destination.*

[†] For the purposes of this thesis, a page means either a page on a piece of paper, or the viewing area of a text window.

There are several ways to specify the placeholder symbol:

- *Specify placeholder symbol for both source and destination:* The user draws the symbol for both halves of the command – *move to star* and *move from star*, similar to the way the annotation is usually made.
- *Specify placeholder symbol for the source only:* After the user draws the placeholder symbol for the first part of the move command, the symbol remains visible in the margin. To specify the destination, the user draws a line from the visible symbol to the destination. Note that the symbol remains visible until the command is completed, even if the user navigates over several pages.
- *Use a standard set of symbols:* Instead of drawing a symbol, the user chooses one from a list of available symbols. The user makes the selection for both the first and second parts of the broken move command.

There are advantages and disadvantages to each of the above. Specifying the symbol for both source and destination requires mark recognition, but is the most similar to what is done in annotating documents. Specifying the symbol once does not require mark recognition, and allows user defined symbols, but keeping it visible clutters up the margin and is likely to cause problems. Using a standard set of symbols from a menu or list also does not require mark recognition, but it does not allow user defined symbols. We chose the last method – using a standard set of symbols, because we were able to design and implement a simple interaction technique for it. The other two methods are left for future work.

In contrast to the other editing commands, the broken move command consists of two separate marks. Therefore, it requires some type of dialogue management. We have designed three possible dialogues that could be implemented:

- *Two distinct move commands:* The first and second halves of the broken move could be treated as "move to placeholder" and "move from placeholder" respectively. In this case the two halves are simple move commands with a special destination or source.
- *Pending broken move – Unforced:* The first half of the broken move remains unprocessed until the user specifies the second half, at which time the entire move command is processed.
- *Pending broken move – Forced:* Similar to the unforced pending move, however, the user cannot specify any intermediate commands except navigation actions.

The *two distinct move commands* dialogue is similar to cut and paste commands, with the exception that the user can use multiple clipboards and specify the identifying symbol associated with each clipboard. With these multiple user-identified clipboards, the power of the broken move command is extended. However, several issues arose in designing and implementing these clipboards. In particular, the clipboards need to be editable, and be made visible upon the user's request. Instead, the forced pending broken move dialogue was incorporated into MATE. Multiple user-defined clipboards appears promising and is left as future work.

### 3.4.3 Object Oriented Approach

The main problem with incorporating annotations is that the annotations apply to a specific version of a document, but incorporating applies them to the current version of that document. Therefore, the key to incorporating annotations is in translating an editing command from one version of the document to another version.

In order to accomplish this translation there must be enough information linking the two versions. Most text editors base their commands on the locations of text and on strings stored in buffers, such as "paste whatever is in the first buffer after line n". This approach will not work for MATE as the locations of text in the current version are constantly changing with respect to the text in the original annotated version. Figure 3.5 shows an example of why using the location of the text will not work.

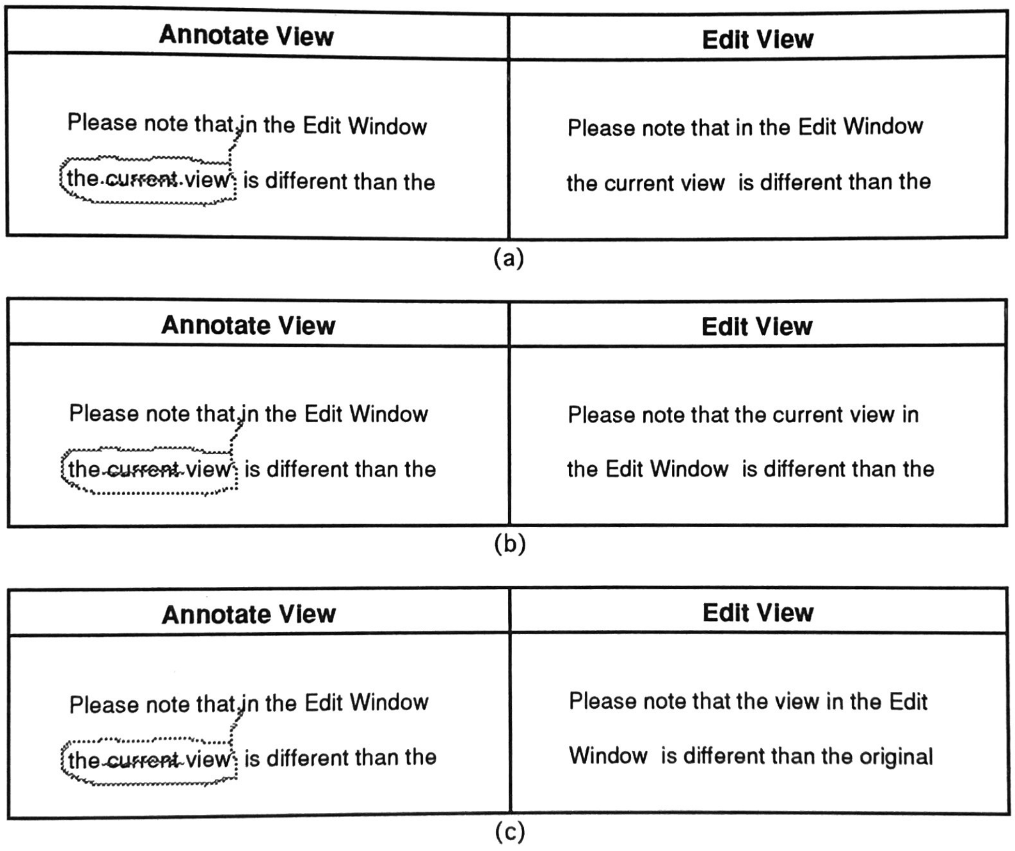
Our solution is to treat each character as a text object. By assigning the following attributes to each text object, a translation between the two versions of a document can be made:

- the letter or symbol of the character,
- the character's location in the original, annotated version of the document,
- the character's location in the current, edited version of the document.

Other attributes are needed to handle the translation of specific commands. For example, objects are never deleted, they are only marked for deletion by setting the *deleted* attribute. These additional attributes are identified and described in Chapter 4.

Using the object-oriented approach, MATE can handle the above example in Figure 3.5 The move command in (a) changes the current character position attribute of the moved text objects and any text objects that follow it. The delete command in (b) finds the text objects associated with *current*. These objects are then marked as deleted and will not be displayed in the current version.

As will be shown in the next sections, our object-oriented approach enables powerful Do / Undo / Redo mechanisms – Section 3.4.4, and provides straightforward tests for conflicts among commands – Section 3.4.5.


Annotate View | Edit View

Please note that in the Edit Window the ~~current~~ view is different than the | Please note that in the Edit Window the current view is different than the

(a)

Annotate View | Edit View

Please note that in the Edit Window the ~~current~~ view is different than the | Please note that the current view in the Edit Window is different than the

(b)

Annotate View | Edit View

Please note that in the Edit Window the ~~current~~ view is different than the | Please note that the view in the Edit Window is different than the original

(c)

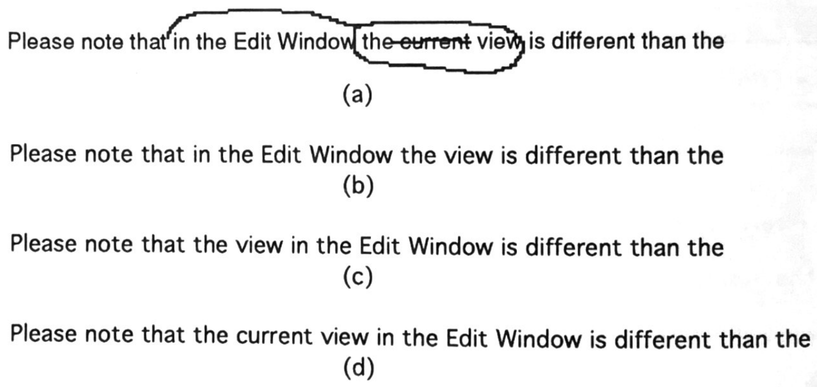
**Figure 3.5:** The changing relationship between the Annotation and Edit Views *The two views start out the same in (a), but as commands are executed, move in (b), delete in (c), the Edit View changes. Note that the deletion of* current *must be independent of* current's *location in the Edit View.*

### 3.4.4 Do / Undo / Redo

To select an annotation for incorporation, the user simply taps on the annotation with the stylus or pointer. A logical extension is to use this same tapping interaction for Undoing annotations which have already been incorporated. To make the interaction work, the underlying text editing mechanisms must be properly designed. This design must satisfy the following requirements:

- *Order Independence:* The effects of a set of commands should be independent of the order in which they are incorporated. For example, in Figure 3.6, the result (c) should occur irregardless of the order of incorporating the move and delete commands.

- *Undo equals the Inverse of Do:* All commands must have an inverse, and this inverse command can be applied at any time†. The results of the inverse command should be the same as if the command was never incorporated at all. For example, in Figure 3.6, the result (d) should occur irrespective of whether the delete command was Done and Undone, or never Done at all.



Please note that in the Edit Window the ~~current~~ view is different than the

(a)

Please note that in the Edit Window the view is different than the

(b)

Please note that the view in the Edit Window is different than the

(c)

Please note that the current view in the Edit Window is different than the

(d)

**Figure 3.6:** Doing and Undoing annotations independent of order
*The initial text is shown in (a). (b) is the result after the deletion is performed. (c) is the result after the move is performed. After undoing the delete command, the desired result is (d).*

These requirements can be satisfied by following two rules:

- the text is treated as set of objects

- commands only modify the attributes of these text objects

For example, the delete command marks the selected text objects as deleted by setting their *deleted* attribute. The inverse of delete simply resets the deleted attribute back to the initial state. The specific implementation of each command is described in Chapter 4.

† This is not entirely true as intermediate commands may cause conflicts with the inverse command. Conflicts are covered in Section 3.4.5.

*Comparison with Undo – Skip – Redo*

Our Do and Undo mechanisms are functionally equivalent to Undo, Skip and Redo (Vitter, 1984), but the interactions needed to achieve the same result are very different. This can be demonstrated by the following example:

Say the user has already done the following:

- delete the word "current" in Figure 3.6(a), resulting in Figure 3.6(b)
- move the text "the view" resulting in Figure 3.6(c)

and wants to undo the delete command resulting in Figure 3.6(d).

With the Undo, Skip, and Redo method, the user would go through the following process:

- Undo the move,
- Undo the delete,
- Skip the delete,
- Redo the move.

In contrast, with our Do / Undo mechanism, the user would simply select the delete annotation to Undo it.

*Unlimited Undo / Redo Last*

An additional benefit of the Do / Undo by selection mechanism, is the capability for an unlimited Undo / Redo last command mechanism. With the requirement that all commands must have an inverse, and by keeping a history of the commands performed, MATE has sufficient information to undo all of the commands back to the beginning of the editing session. Very little information is required to store each command in the history list, allowing the list to be virtually unlimited in length.

There are two important differences between the Do / Undo mechanism and the Undo / Redo Last mechanism:

- Undo / Redo Last can work both in Incorporation Mode and in Edit Mode, whereas Do / Undo can only work in Incorporation Mode.
- Do / Undo has the equivalent functionality to Undo, Skip, and Redo. Undo / Redo Last would require the Skip capability to match this functionality. This capability has not been designed nor implemented into the current version of MATE.

### 3.4.5 Conflicting Commands

As the document, changes from its original version there may be situations in which an annotation is ambiguous or is no longer applicable to the current version of the document. These situations are termed *conflicts* and may be any of the following:

- the annotation is ambiguous or meaningless to the user
- the annotation is ambiguous or meaningless to MATE
- MATE's interpretation is different than the user's

The ideal situation is that MATE and the user are in agreement with the meaning of an annotation, and find the same situations ambiguous or no longer applicable. By treating the text as objects, we hope to achieve this situation. To accurately determine how close MATE is to our ideal would require large scale user testing, which is beyond the scope of this thesis.

In terms of the design of MATE a conflict can be defined as a situation in which a command cannot be translated from the original version of a document to the current version. An example of such a situation is shown in Figure 3.7.

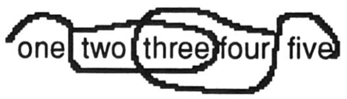


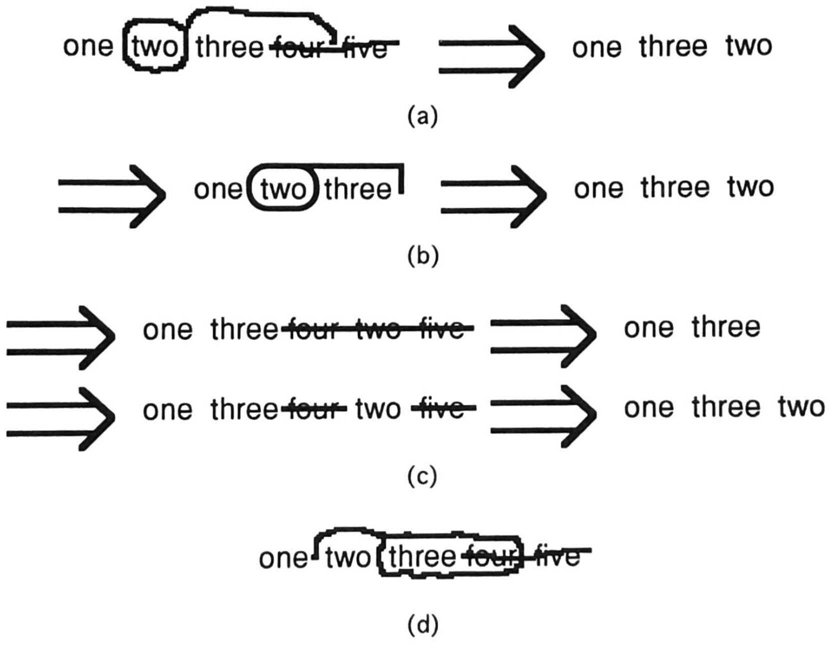
**Figure 3.7:** Conflicting commands
*Once either of the move commands is incorporated, the meaning of the other becomes ambiguous and can no longer be translated into a meaningful command.*

*Order Dependency and Compound Commands*

Certain pairs of annotations seem to function together. For example, in Figure 3.8(a), the move and the delete command are really meant to work as a *replace* command. Such command pairs are termed *compound commands*, as they are formed from two or more simple commands.

If we try to execute the compound command by executing each of its constituent commands, we notice that the order of execution is important. In our example in Figure 3.8, the replace command works if the delete command is performed first and the move command second (b), but it does not work if the order is reversed (c)†. However, as commands cannot be order dependent, (b) cannot be allowed if (c) is not allowed.

The solution is to incorporate the entire compound command at once. But in order to do so, we must solve the issues of syntactic segmentation mentioned in Section 2.2.2. In particular, MATE must be able to identify and recognize compound commands, and also be able to distinguish a selection of the compound command from a selection of one of its constituent commands. In the current implementation of MATE, compound commands are treated as conflicts; the design and implementation issues concerning them are left as future work.



one two three ~~four five~~ ⇒ one three two
(a)
⇒ one two three ⇒ one three two
(b)
⇒ one three ~~four two five~~ ⇒ one three
⇒ one three ~~four~~ two ~~five~~ ⇒ one three two
(c)
one two three ~~four five~~
(d)

**Figure 3.8:** Order dependency and compound commands
*The result of the delete and move commands depends upon the order in which they are executed. In (a) they are treated as a compound* replace by move *command. In (b), the delete is executed first, followed by the move command, yielding the same result as the replace command. But if the move is done first as shown in (c), the delete command becomes ambiguous. Part (d) shows why (c) should be considered ambiguous,. If the move command is performed first in (d), it also arrives at situation (c), but (d) is an ambiguous situation to start with and therefore (c) must be considered ambiguous. The only solution is to treat the delete and move commands in (a) as a compound command.*

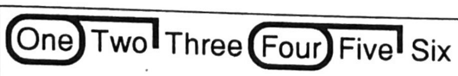
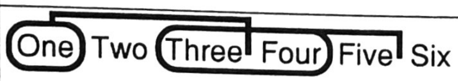
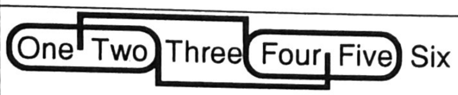
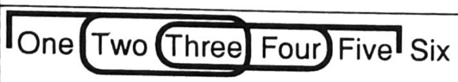
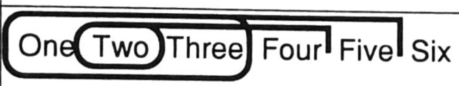
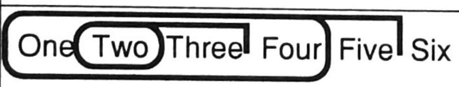
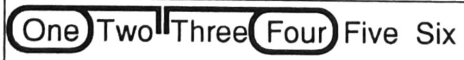
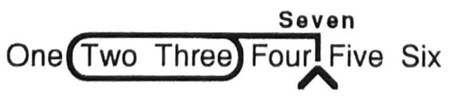
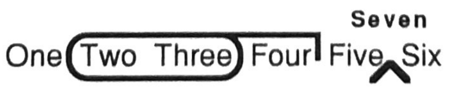
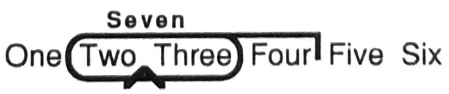
† Figure 3.8(d) illustrates why (c) should not work; (d) is clearly an ambiguous situation, but if (c) is allowed to work as a replace command, then (d) should also be allowed.

*Determining Conflicts*

It is sufficient to test for conflicts between two markings, because it would never be the case that there is a conflict between three or more markings without there being a conflict between at least one pair out of the these markings. The following rules are used in determining whether an annotation conflicts with previously incorporated annotations:

- *Delete:* A delete command acts on a connected sequence of text objects. A conflict arises if this sequence is broken, or if any part of this sequence has already been deleted.
- *Move:* The source of a move command is specified by its starting and ending text objects; any action can be applied to the text objects between these two text objects. The destination is specified by a single text object. A conflict occurs under any of the following conditions:
  - Either of the two source boundary objects is moved independent of the other;
  - A delete command crosses over either or both of the source boundaries;
  - The destination is within the source;
  - The destination has been deleted;
  - Text objects have already been placed in the destination, either by another move command, or by an insert.
- *Insert:* An insert command is similar to the destination of a move command. A conflict occurs if the insert location has been deleted, or if text objects have already been placed in the insert location.
- *Undo Delete:* The undoing of a delete command is similar to the insert command, and has the same rules for conflicts.
- *Undo Move:* The undoing of a move command is another move command with the source and destination reversed.
- *Undo Insert:* The undoing of an insert is similar to a delete command in that it acts on a connected sequence of text objects, but it has different rules for conflicts. A conflict arises if the sequence is broken, or if a delete command crosses over either or both of the boundaries of the inserted text. However, a conflict does not occur if the only text objects deleted are between the boundaries.

Table 3.1 summarizes how these rules apply to pairs of commands.

| Situation | Example | Conflict | Reason | Conflict Rule |
|---|---|---|---|---|
| **Delete – Delete** | | | | |
| no overlap | One ~~Two~~ Three ~~Four~~ Five Six | No | | |
| overlapping | One ~~Two Three Four~~ Five Six | Yes | Nonsense | Part of the sequence has already been deleted |
| embedded | One ~~Two Three Four~~ Five Six | Yes | Nonsense | Part of the sequence has already been deleted |
| **Move – Move** | | | | |
| no overlap |  | No | | |
| move into move |  | No | | |
| transpose |  | Yes | Compound command | The destination of the second move command is within its source. |
| overlapping |  | Yes | Nonsense | One of the boundaries is moved independently of the other. |
| move from within move |  | No | | |
| move within move |  | No | | |
| move to same location |  | Yes | Ambiguous | Text objects have already been placed in the destination. |
| **Insert – Move** | | | | |
| move to same location as insert |  | Yes | Ambiguous | Text objects have already been placed in the destination. |
| no overlap |  | No | | |
| insert within move |  | No | | |

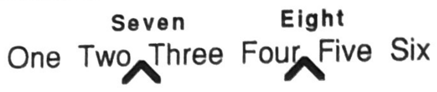
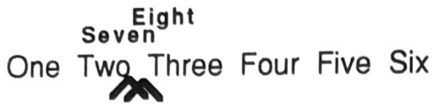
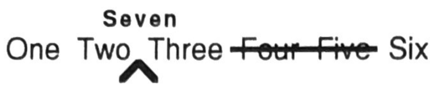
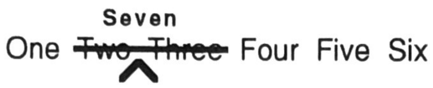
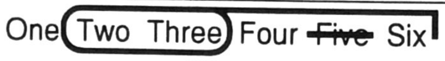
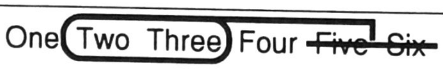
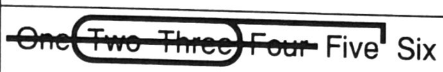
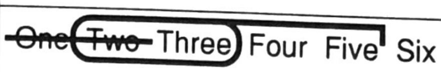
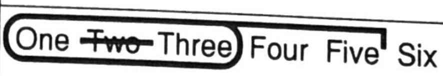
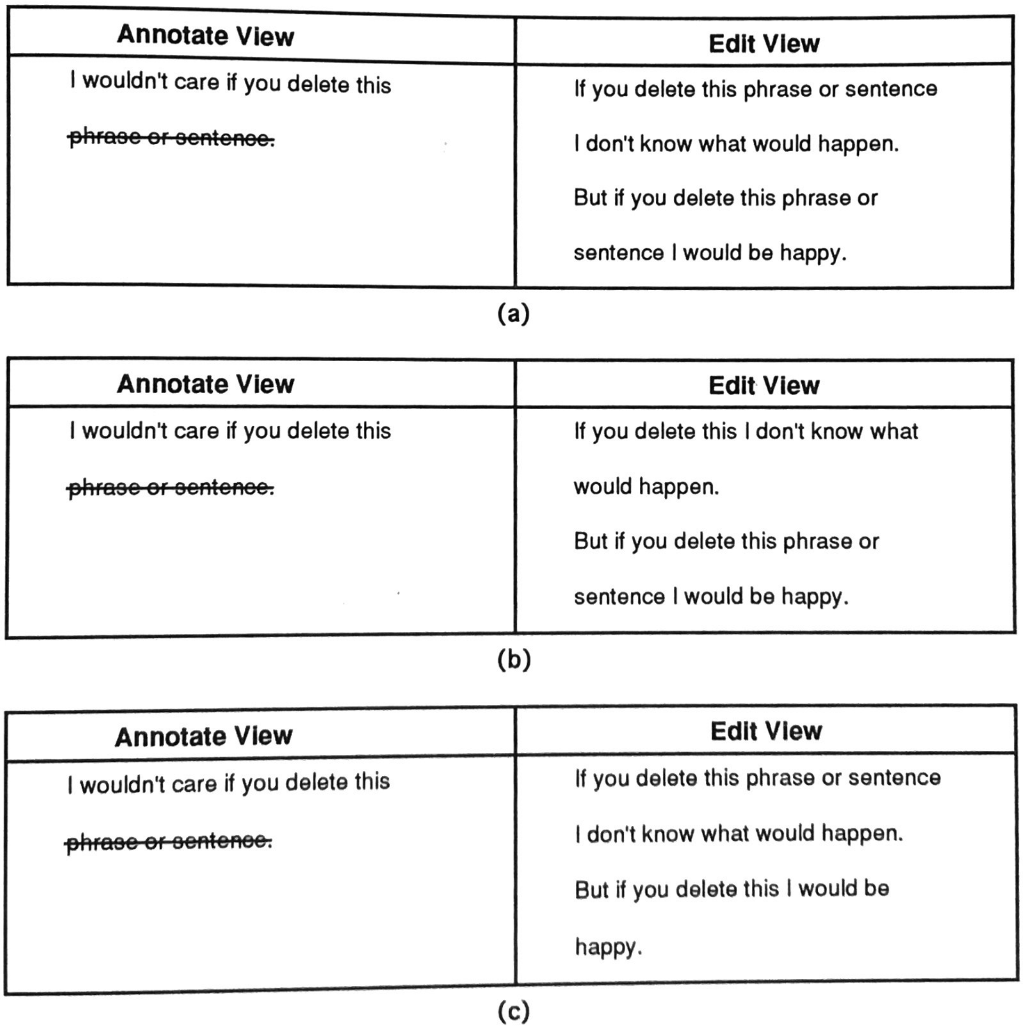
| | | | | |
|---|---|---|---|---|
| **Insert – Insert** | | | | |
| insert at different locations |  | No | | |
| insert at same location |  | Yes | Ambiguous | Text objects have already been placed in the destination. |
| **Delete – Insert** | | | | |
| no overlap |  | No | | |
| replace |  | Yes | Compound command | Delete: The sequence has been broken.<br>Insert: The insert location has been deleted. |
| **Delete – Move** | | | | |
| no overlap |  | No | | |
| replace by move |  | Yes | Compound command | Delete: The sequence has been broken.<br>Move: The destination has been deleted. |
| delete source of move |  | Yes | Ambiguous (move 1st)<br>Nonsense (delete 1st) | Delete: The sequence has been broken.<br>Move: Both boundaries have been deleted. |
| overlap |  | Yes | Ambiguous | Delete: The sequence has been broken.<br>Move: A boundary has been deleted. |
| delete within move |  | No | | |

**Table 3.1:** Conflicting Command Pairs
*All of the possible pairs of the delete move and insert commands are shown. If a conflict exists, a reason from the user's perspective, and the rule which would detect the conflict are given. Note that the rule is stated from the perspective of an attempt at applying the second command after the first command has already been executed. Also note that the first command can either be an incorporated annotation or a direct editing command, but the second command is an incorporated annotation; an editing command never conflicts with previous commands.*

### 3.4.6 Relating the Original and Current Views

The preceding sections have explained how MATE can relate the original and current versions of the document to incorporate annotations, but it is also important to support the user in understanding the relationship between the two versions. In particular, the user may require additional support after incorporating an annotation, or while navigating through the document.



| Annotate View | Edit View |
|---|---|
| I wouldn't care if you delete this ~~phrase or sentence.~~ | If you delete this phrase or sentence I don't know what would happen. But if you delete this phrase or sentence I would be happy. |

(a)

| Annotate View | Edit View |
|---|---|
| I wouldn't care if you delete this ~~phrase or sentence.~~ | If you delete this I don't know what would happen. But if you delete this phrase or sentence I would be happy. |

(b)

| Annotate View | Edit View |
|---|---|
| I wouldn't care if you delete this ~~phrase or sentence.~~ | If you delete this phrase or sentence I don't know what would happen. But if you delete this I would be happy. |

(c)

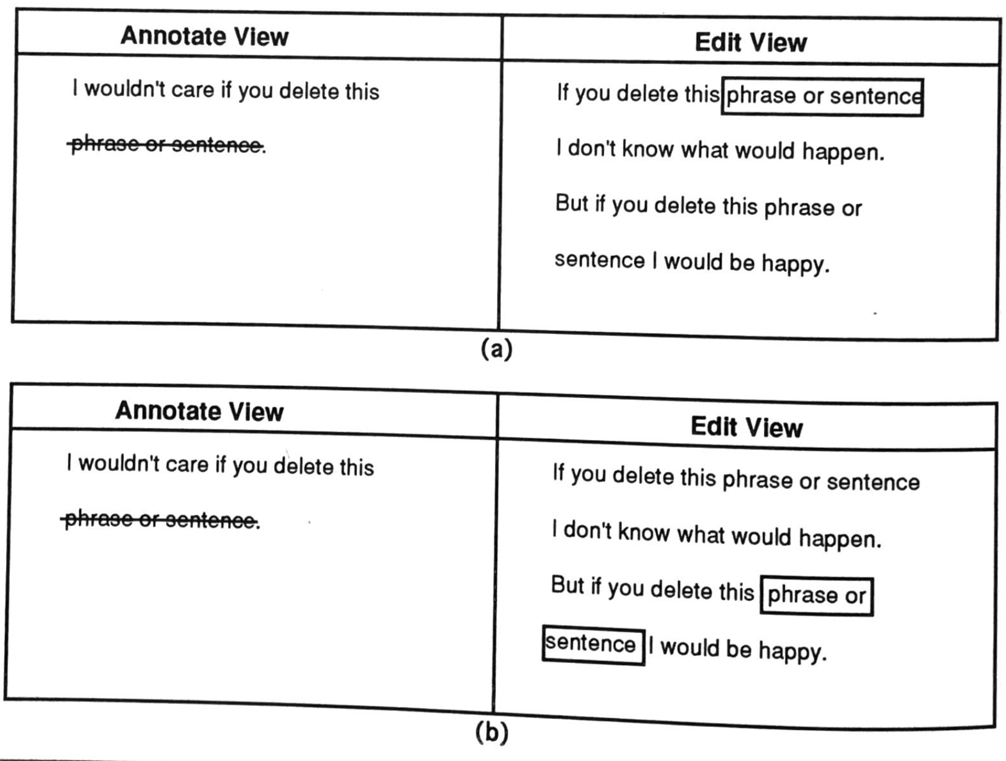
**Figure 3.9:** "Delete which phrase?" An example of possible user confusion
*After many editing operations the two views appear as shown in (a). If the user then selects the delete mark for incorporation, it is unclear to the user what MATE will do. (b) & (c) show two possible results, depending upon how the document reached its current state.*

*General Support for Relating the Two Views*

The general relationship problem is illustrated in Figure 3.9. After many editing operations, the actual meaning of the delete annotation is ambiguous to the user. Some support mechanism is necessary to provide the user with more information.

There are several solutions to this problem:

- *Update the original view:* One method to help relate the two views is to allow the user to update the Annotation View to the state of the Edit View. But this has the same problems as the single view alternative described in Section 3.3.
- *Highlight the text associated with the command:* Another method is to highlight the text associated with the annotation. If, in Figures 3.9 and 3.10, the delete annotation applied to the first phrase, the text would be highlighted as shown in Figure 3.10(a). If, however, the delete annotation applied to the second phrase, the text would be



| Annotate View | Edit View |
|---|---|
| I wouldn't care if you delete this ~~phrase or sentence.~~ | If you delete this phrase or sentence I don't know what would happen. But if you delete this phrase or sentence I would be happy. |

(a)

| Annotate View | Edit View |
|---|---|
| I wouldn't care if you delete this ~~phrase or sentence.~~ | If you delete this phrase or sentence I don't know what would happen. But if you delete this phrase or sentence I would be happy. |

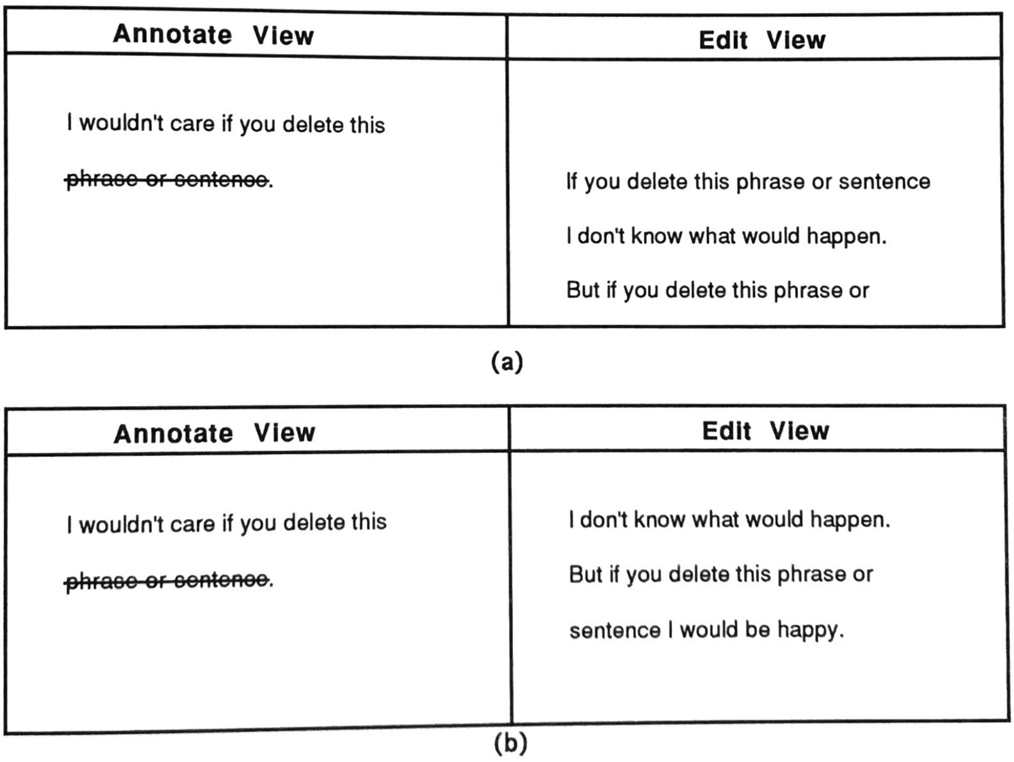
(b)

**Figure 3.10:** Highlighting Text

*If the delete annotation applied to the first phrase, the text would be highlighted as shown in (a). If, however, the delete annotation applied to the second phrase, the text would be highlighted as shown in (b).*

highlighted as shown in Figure 3.10(b).

- *Do / Undo the annotation:* As annotations can be done and subsequently undone, the user can simply incorporate an annotation, examine its effects, and then undo it.
- *Line-up the two views according to the text of interest:* A final method is to line-up the two views of the document along a specific piece of text. For example, in Figure 3.11, if the user selected the word *phrase* in the Annotation View, the Edit View would be adjusted so that the corresponding *phrase* is aligned with that in the Annotation View. An advantage of lining-up versus Highlighting or Do / Undo is that it does not require an annotation that MATE can incorporate. Lining-up depends only on the text.



**(a)**

| Annotate View | Edit View |
|---|---|
| I wouldn't care if you delete this | |
| ~~phrase or sentence~~. | If you delete this phrase or sentence |
| | I don't know what would happen. |
| | But if you delete this phrase or |

**(b)**

| Annotate View | Edit View |
|---|---|
| I wouldn't care if you delete this | I don't know what would happen. |
| ~~phrase or sentence~~. | But if you delete this phrase or |
| | sentence I would be happy. |

**Figure 3.11:** Aligning the two views
*If the user selects the word "phrase" in the Annotation View, the Edit View would be adjusted so that the corresponding* phrase *is aligned with that in the Annotation View. (a) shows the result if the first "phrase" in the Edit View corresponds with that in the Annotation View; (b) shows the result if the second "phrase"* corresponds.

### *Multiple Points of Interest and Incorporation Support*

At any point in the editing process, the user may be interested in several pieces of text which have been separated from each other. In particular, this occurs when incorporating an annotation, as shown in Figure 3.12. When the move annotation is incorporated, the user is only able to see either the source text being inserted into the destination, or the source text being removed from its original position, but not both.

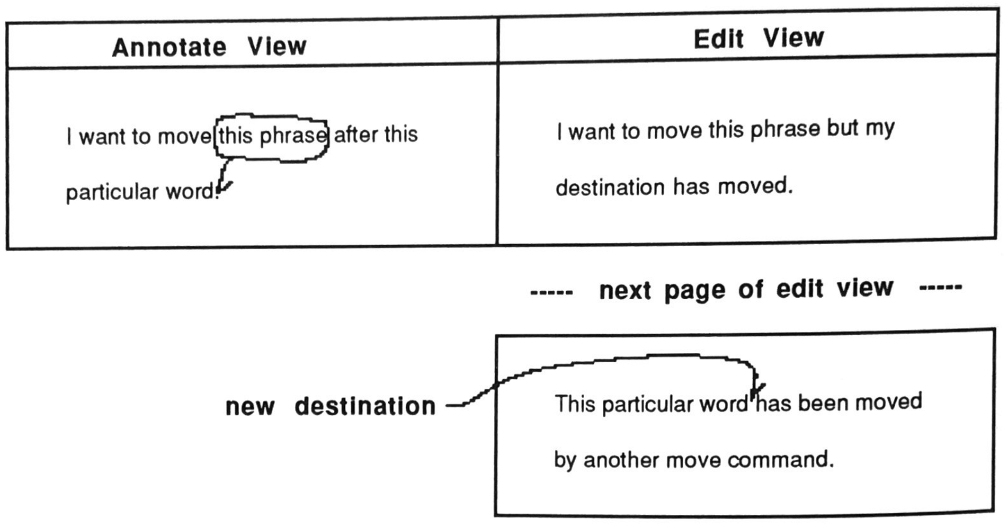

Annotate View
Edit View
I want to move this phrase after this particular word.
I want to move this phrase but my destination has moved.
----- next page of edit view -----
new destination
This particular word has been moved by another move command.

**Figure 3.12:** Separated points of interest
*The source and destination of the move command are on the same page in the Annotation View, but are on different pages in the Edit View. This causes problems, because the user cannot see all of the results of the command.*

Two questions arise from the above problem:

1. How should the Edit View appear after the command is incorporated?

   This can be broken down into the following set of questions:

   - Should the Edit View retain its current position?
   - Should the Edit View show at least one of the points of interest?
   - Is a single Edit View insufficient?

   These questions are partly answered by the user testing in Chapter 5. More complete answers will require a more in depth study.

2. How can the user receive the appropriate feedback for all points of interest?

   Two possible solutions are *jumping to a point of interest* and *multiple sub-views,* discussed below.

- *Jumping to a point of interest:* This solution allows the user to quickly *jump* to a point of interest in one window by specifying the corresponding point of interest in the other window. This is exactly the same mechanism as lining-up the two views mentioned above. Figure 3.13 illustrates how jumping can provide feedback to the user.

- *Multiple Sub-views:* In this solution to the multiple points of view problem, additional edit and annotation windows, displaying the document at various positions, are shown. After the move annotation, from the example in Figure 3.12, is incorporated, two edit windows would be displayed, as shown in Figure 3.14. The upper window shows the initial position of the source text, and the lower window shows the destination.

  This solution of having multiple sub-views is an extension to having one window for the Annotation View and one for the Edit View. One problem with this solution is that the screen may become very cluttered. Also several of the other problems in

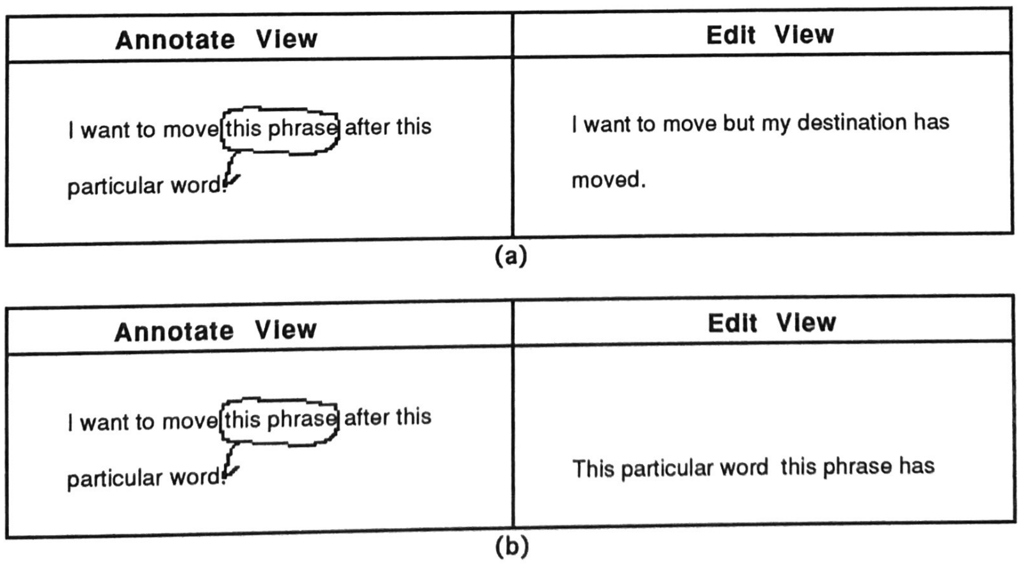

Annotate View
Edit View
I want to move this phrase after this particular word.
I want to move but my destination has moved.
(a)
Annotate View
Edit View
I want to move this phrase after this particular word.
This particular word this phrase has
(b)

**Figure 3.13:** Jumping to points of interest

*In Figure 3.12 the source and destination of the move command are on the same page in the Annotation View, but are on different pages in the Edit View. By selecting "I want to move" in the Annotation View, the Edit view jumps to the position shown in (a), which can show the removal of "this phrase". By pointing to "particular word" in the annotate view, the Edit view jumps to the position shown in (b), which can show the insertion of "this phrase" into the destination.*

relating two views become magnified with additional windows. In this thesis, one Annotation View and one Edit View are used. The extension to multiple sub-views is left for future work.

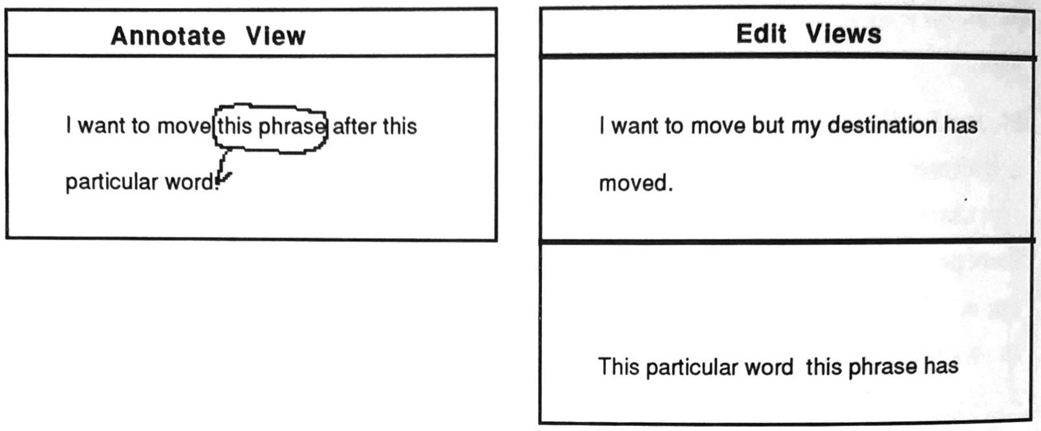

Annotate View
I want to move this phrase after this particular word!
Edit Views
I want to move but my destination has moved.
This particular word this phrase has

**Figure 3.14:** Multiple Edit Views
*By showing multiple sub-views of the document, the user can see all of the effects of a command at once.*

*Navigation Support*

Navigation has similar problems to those of incorporating annotations; it may be unclear what to show in the Edit View while the user navigates in the Annotate View and vice versa. This is illustrated by the example in Figure 3.15.

The first issue is whether each view is dependent or independent of the navigation in the other view. We can assume that the user will want both situations at different times. To allow for this both independent and dependent navigation mechanisms are implemented in MATE.

The second issue is how should the dependent navigation work. There are two alternatives:

- *Content dependent:* The dependent view aligns itself so that a portion of the text is the same in both views. An example is to align the dependent view so that the text in the top line of both views is the same. In Figure 3.15, a page down in the Annotation View would cause the Edit View to adjust so that paragraph 3 is at the top of both views.
- *Relative movement dependent:* The dependent view moves the same distance as the

view being navigated in. In Figure 3.15, a page down in the Annotation View would cause a page down in the Edit View, so that paragraph 1 is at the top of the Edit View.

The best method of determining what text the user is interested in is by allowing the user to select it. Therefore, the jumping to a point of interest and aligning the views mechanism is the only content dependent navigation command in MATE. All other dependent navigation commands are relative movement dependent.

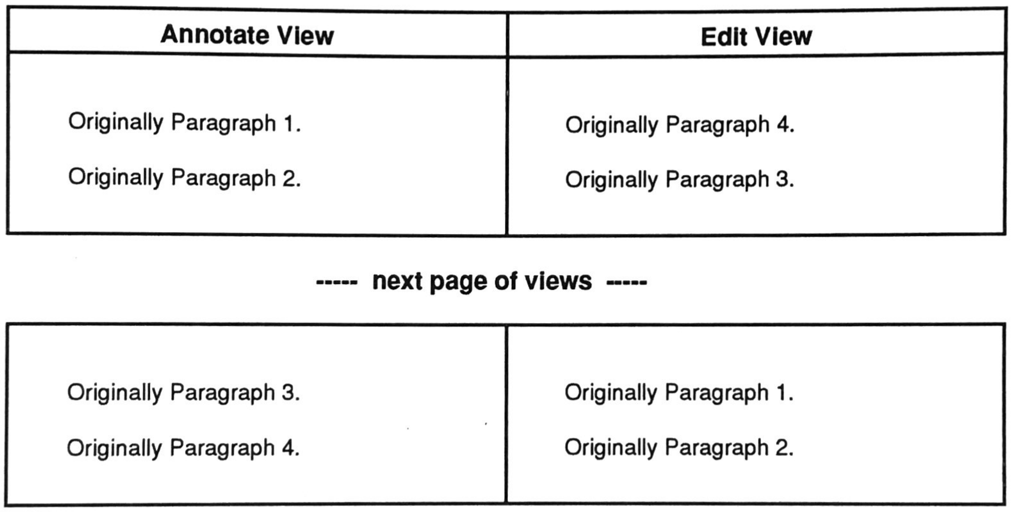


**Figure 3.15:** Navigation problems between Annotation and Edit Views
*If the user pages down in the Annotation View, what should happen in the Edit View? There are several possibilities, including: remain the same, page down, and align the top line of the Edit window with the top line of the Annotation window. But the answer is unclear.*

## 3.5 Summary

MATE is designed to satisfy the following four requirements:

- To allow users to annotate text with markings.
- To accept text-editing commands in the form of markings which are visually distinct and consistent with the markings used for a corresponding annotation.
- To achieve a smooth transition from using marks as annotations and as editing commands.
- To allow users to view annotations in a legible manner from several sources.

These requirements map into three main operations: Annotating, Editing, and Incorporating. Therefore MATE is designed to have three modes, one for each operation. The designs of the Annotation and Edit Modes are relatively straightforward, but several design alternatives for Incorporation Mode were examined. From these, a *two view* design, which integrates the Annotation and Edit Modes, was chosen. From observations in Chapter 2, we can also term Annotation Mode as Deferred Mode, and Edit Mode as Immediate Mode. Incorporation Mode, with the two view design, can be thought of as a *Combined Mode*.

To support multiple sets of annotations, each set of annotations is colour-coded and stored separately from the other sets of annotations and the text. To allow annotations to be incorporated as editing commands, annotations are stored as sequences of events.

MATE has three editing commands: *delete*, *move*, and *insert*, shown in Figure 3.3. In addition a special case of the move command, *broken move with placeholder symbols*, was examined, Figure 3.4. Depending upon the manner in which they are implemented, placeholders can be used as multiple user-defined clipboards. However, we chose a simpler design for this thesis.

The main problem of incorporating annotations is in translating editing commands from one version of the document to another. This problem is solved by using an object-oriented approach. In this solution, each character is a text object with attributes., and commands only modify the attributes of these text objects.

This object-oriented approach allows commands to be undone by selection in addition to being done by selection. The Do / Undo mechanism is just as powerful and easier to use than an Undo, Skip, Redo mechanism. An additional benefit from our Do / Undo mechanism, and our object-oriented approach is the capability of having an Undo / Redo last mechanism which needs to store only a minimal amount of information.

Conflicts may arise as there may not always be a valid translation of a command from one version of the document to another. To determine when these conflicting situations occur, a set of rules was developed.

In addition to providing a solution for the translation problem, support for the user in understanding the relationship between the two versions is also important. Several solutions were examined:

- Updating the original view
- Highlighting the text associated with a command
- Doing / Undoing annotations
- Aligning the two views according to a section of text, which can also be thought of as jumping to a point of interest
- Multiple sub views
- Dependent and independent navigation

All of these have been implemented into MATE, except for updating the original view and multiple sub-views.

Several of the benefits mentioned in this chapter were not originally intended in the design of MATE, but resulted from searching for robust solutions which met the design requirements. Many of these benefits and solutions are new and unique. They are not necessarily better than existing solutions, but give rise to new possibilities and different ways of thinking about editing and annotating documents and the use of markings.

# Chapter 4

# Implementation

In the previous chapter we developed the general design of MATE. Some aspects of MATE, such as the interface and navigation mechanisms, require an iterative approach to their design, and are covered in this chapter, along with a description of the actual implementation.

In this chapter we describe the following:

- the general layout, design, and implementation of MATE's interface,
- the functions and the interface to support the handling of annotations,
- the implementation of the editing commands,
- the functions and the interface to support the incorporation of annotations, and
- the design and implementation of the navigation commands.

Several implementation issues, not directly related to our research, needed to be addressed in order to build a functional system. These include recognition techniques and utility functions such as file handling, and are discussed only when they affect our central research questions.

## 4.1 General Interface Design and Implementation

MATE was developed on a SUN SPARCstation, with a 16 inch colour monitor, in the C programming language, and the X windowing system. The primary input device is a Pencept graphics tablet and stylus. The Pencept pen is a ball-point pen which leaves ink on paper. It also has a button near the tip as shown in Figure 4.1. The graphics tablet is approximately one and a half feet square with paper overlaid on top of it.

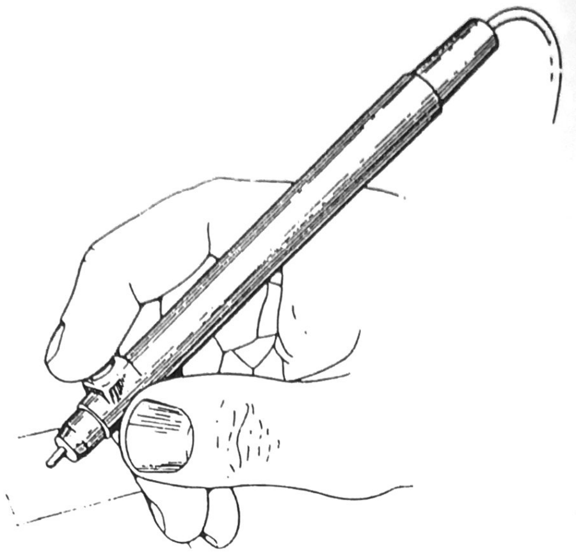


**Figure 4.1:** The Pencept Stylus (Pencept, 1986)

*The MATE Window*

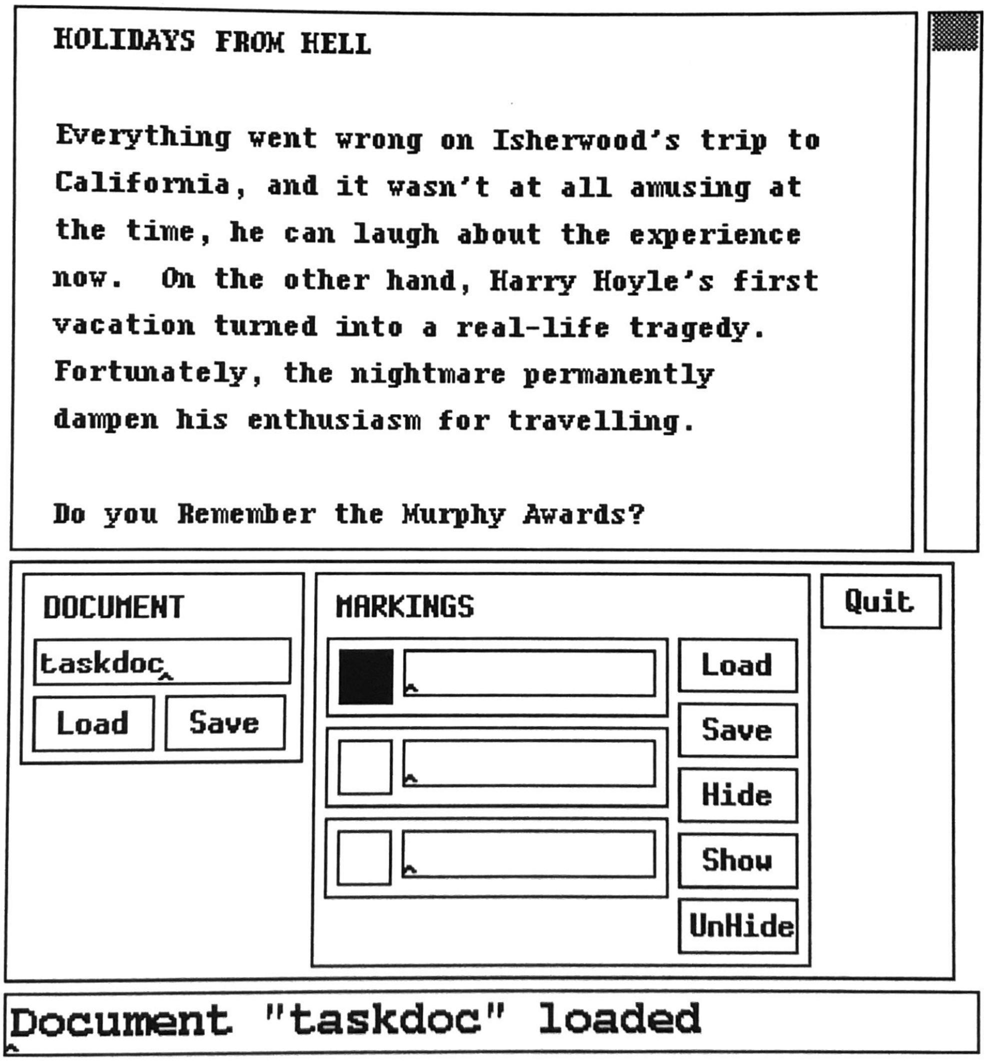
MATE has a slightly different interface for each of its three modes. Annotation Mode is shown in Figure 4.2, Edit Mode in Figure 4.3, and Incorporation Mode in Figure 4.4. In all three modes, the document is shown at the top; below the document is an area for utility buttons, file handling, and a message line. Our primary focus is on the interactions within the document area; the remainder of the graphical user interface received less attention in its design.

The layout of text in the document area is constrained by several factors. The screen size places a maximum restriction on the size of the document area. Markability – Section 2.2.1, places minimum restrictions on the line spacing and font size used. But the user needs to see a minimum number of characters and lines in order to work effectively. Between 18 and 25 lines of text can be displayed with 30 to 60 characters per line. For the user testing an 18 point font was used with double spacing; this allowed 18 lines of text with 30 characters per line to be displayed.

*Marking Menus*

To specify actions on the annotations, and for navigation, marking menus are used. Marking menus are an interface mechanism developed by G. Kurtenbach (1991b) which combines popup pie menus with a marking-based approach to specifying commands. They can be used in one of the following ways:

- The user holds the pen down and waits for the pie menu to appear. A selection can be made by moving the pen to the appropriate menu item and lifting the pen.
- The user does not wait for the menu to be displayed. Instead, the user can simply draw a line in the same direction as the menu item and then lift the pen to make the


HOLIDAYS FROM HELL

Everything went wrong on Isherwood's trip to California, and it wasn't at all amusing at the time, he can laugh about the experience now. On the other hand, Harry Hoyle's first vacation turned into a real-life tragedy. Fortunately, the nightmare permanently dampen his enthusiasm for travelling.

Do you Remember the Murphy Awards?

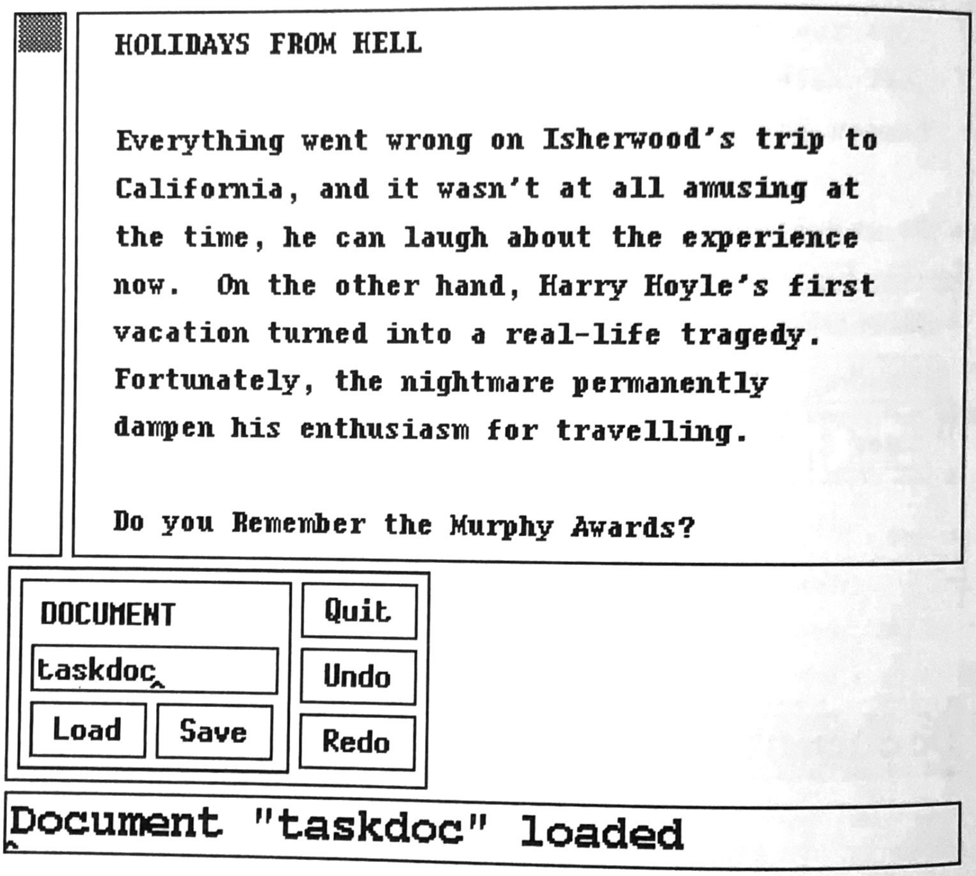
Document "taskdoc" loaded

**Figure 4.2:** The Interface for Annotation Mode
*The document is displayed in the top portion of the window. Markings made in this portion remain visible and can be stored and retrieved. Below the document portion is the file handling portion of the interface. The left side of this portion deals with the text document file, and the right side handles the files for each set of markings. Up to three markings files can be used at once, each with its own colour. Each set of markings can be hidden or shown to allow the markings to be more legible when several sets of markings are in use.*

selection.

Novice users learn the marks by waiting for the menu, while expert users can use the faster method of marking without waiting for a menu.

MATE has three different marking menus, one each for navigation (Section 4.5), annotation handling (Section 4.2), and for specifying placeholder symbols (Section 4.3). Each of these menus is invoked within a specific context and is described in its respective section.



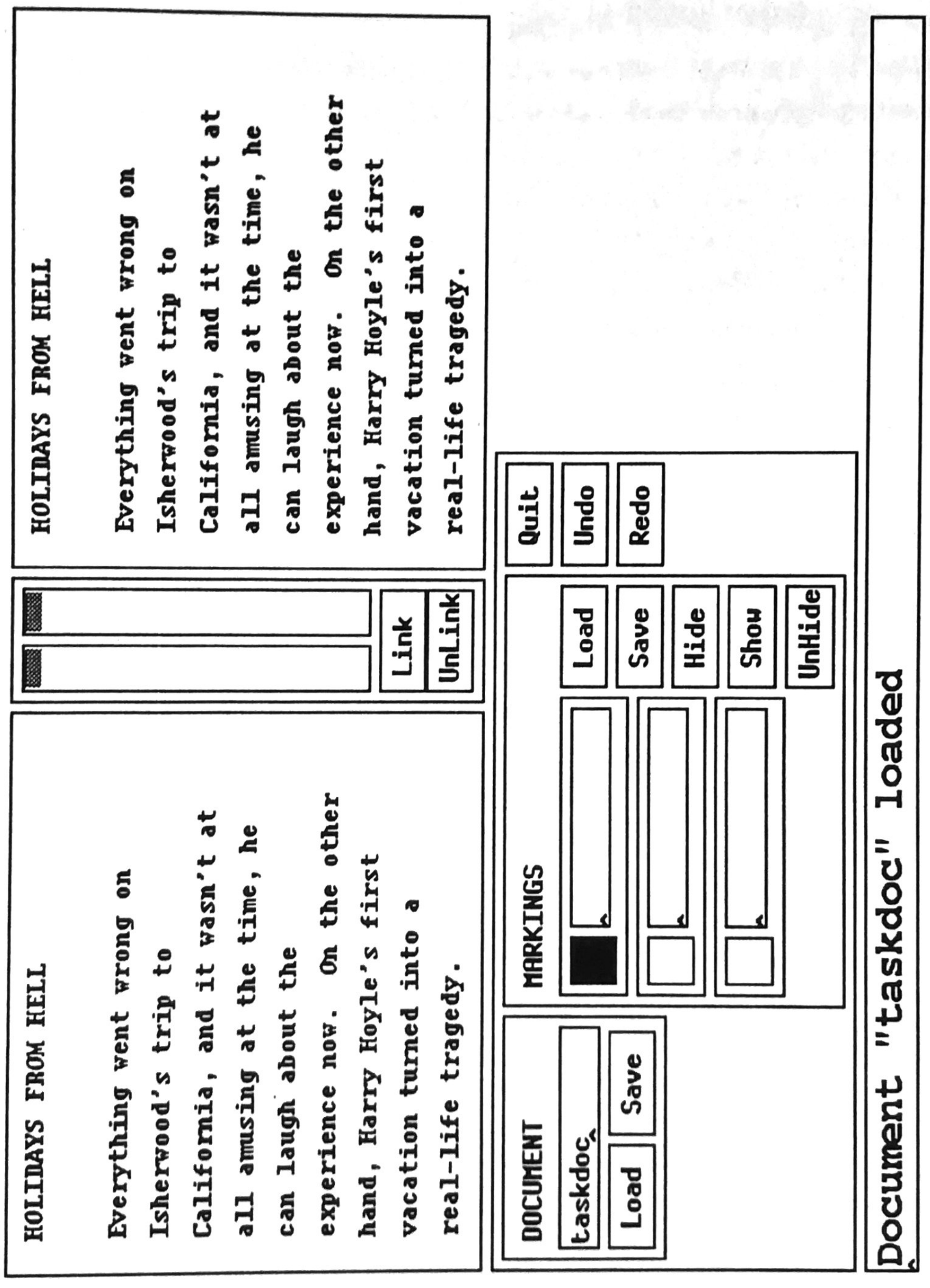
HOLIDAYS FROM HELL

Everything went wrong on Isherwood's trip to California, and it wasn't at all amusing at the time, he can laugh about the experience now. On the other hand, Harry Hoyle's first vacation turned into a real-life tragedy. Fortunately, the nightmare permanently dampen his enthusiasm for travelling.

Do you Remember the Murphy Awards?

Document "taskdoc" loaded

**Figure 4.3:** The Interface for Edit Mode
*The document is displayed in the top portion of the window. Markings made in this portion are immediately interpreted into editing commands. Below the document portion is the file handling portion of the interface. There are also buttons for Undo and Redo last command, mentioned in Section 3.4.4.*



HOLIDAYS FROM HELL

Everything went wrong on Isherwood's trip to California, and it wasn't at all amusing at the time, he can laugh about the experience now. On the other hand, Harry Hoyle's first vacation turned into a real-life tragedy.

HOLIDAYS FROM HELL

Everything went wrong on Isherwood's trip to California, and it wasn't at all amusing at the time, he can laugh about the experience now. On the other hand, Harry Hoyle's first vacation turned into a real-life tragedy.

DOCUMENT

taskdoc

Document "taskdoc" loaded

**Figure 4.4:** The Interface for Incorporation Mode

*The interface for Incorporation Mode can be thought of as placing the Edit Mode interface to the right of the Annotation Mode interface. The additional interface mechanisms for incorporating annotations , such as tapping on a mark to do it, do not affect the graphical layout of the interface and are discussed in Section 4.2.*

## 4.2 Annotation Support

Annotations are made by marking in the document area of the Annotation View, either in Annotation Mode or Incorporation Mode. In addition to incorporating an annotation by selecting it with a tap and unincorporating it with a second tap, other actions can be applied to an annotation via the marking menu shown in Figure 4.5. The Erase function permanently removes a mark, whereas the Hide function temporarily hides the mark. All hidden marks can made visible again with the UnHide button. The Goto and Goback functions are part of the navigation system, and are discussed in Section 4.5.

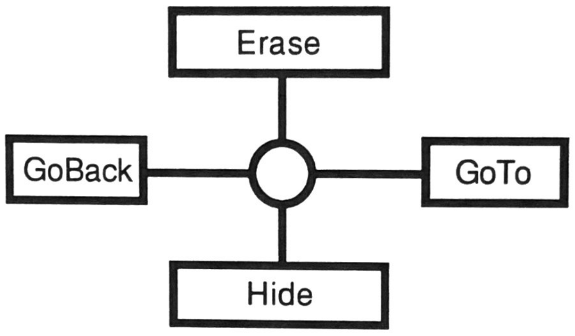


**Figure 4.5:** Annotation Marking Menu
*The annotation marking menu consists of the* ***Erase***, ***Hide***, ***Goto***, *and* ***Goback*** *functions. Erase permanently removes a mark, whereas Hide makes the mark invisible until it is reset back to a visible mark. Goto and Goback are special navigation actions and are covered in Section 4.5.*

The selection and marking menu mechanisms are integrated as a modified version of the marking menu interaction. This interaction is illustrated in Figure 4.6. Note that marking menus normally treat a tap as no selection; our modification distinguishes between a tap – no menu appears, and a non-selection – menu appears.

*Handling Annotations from Several Sources*

MATE supports up to three sets of annotations. This achieved in the markings area of the MATE window, by providing a colour selection box and a filename for each set of markings. To set the active marking set the user clicks on that marking set's colour box. There are five support functions for the active marking set:

- *Load:* Loads the active marking set from filename.
- *Save:* Saves the active marking set to filename.

- *Hide:* Hides all of the marks in a marking set. As was noted in Chapter 3, the annotations can become very cluttered. The problems of clutter are magnified when there are several sets of annotations from different sources. The Hide function allows the user to temporarily remove some of the marks from view in order to reduce clutter.
- *Show:* Undoes the effects of Hide.
- *UnHide:* Undoes the effects of all of the individual Hides from the annotation marking menu.

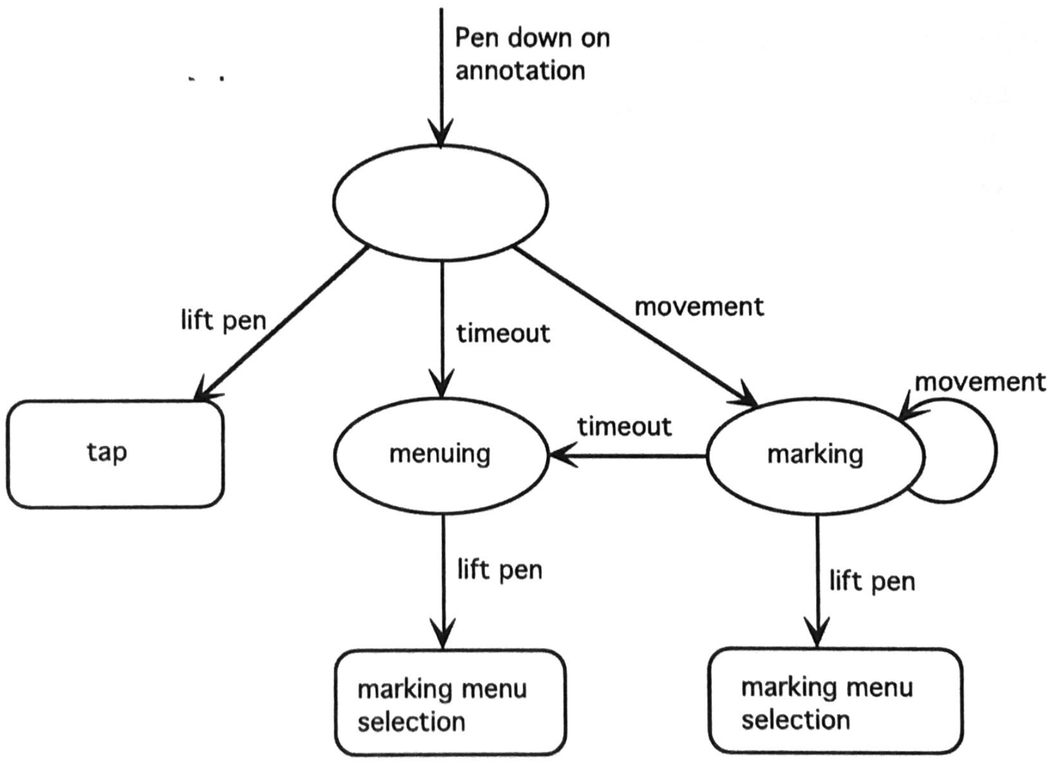


**Figure 4.6:** State diagram for the annotation interaction
*The interaction begins when the user points to an annotation. If the user lifts the pen before the menu time-out period (~ 0.3 seconds), and without drawing a mark, the result is a tap. Otherwise, the regular marking menu interaction occurs.*

## 4.3 The Editing Commands

The marking set for the editing commands that was actually implemented was constrained by the recognition technology available, in addition to being constrained by our design decisions. The marking recognizer can only interpret single strokes; therefore all marks in the marking set are a single line. No character recognizer was available to us at the time of implementation, instead the keyboard is used to enter any text.

The implementation of the editing commands is based upon the design decisions that incorporated annotations are order independent, all commands have an inverse, the text is treated as a set of objects, and commands only modify the attributes of these text objects. The following are the basic attributes of a text object:

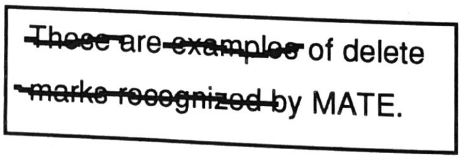
- An *ASCII character value*, depicting the letter or symbol of the character object.
- The *original position in the document*, stored as paragraph number and character position within the paragraph.
- The *original position in the Annotation View*, stored as the line number and character position within the line. This is different than the position in the document as a single paragraph may be several display lines long. This distinction is made because identifying a text object is sometimes easier using the position in the document, and sometimes easier with the position in the viewing window.
- The *current position in the document*, stored as paragraph number and character position within the paragraph.
- The *current position in the Edit View*, stored as the line number and character position within the line.

Other attributes will be added as each command is discussed in detail.

### 4.3.1 Delete

*Interface*

The delete mark is a horizontal line through the text to be deleted. Figure 4.7 shows an example.



~~These~~ are ~~examples~~ of delete
~~marks recognized~~ by MATE.

**Figure 4.7:** Delete marks recognized by MATE
*A delete mark is a single horizontal line through the text to be deleted.*

*Representation*

The delete command can be represented as:

```
delete {list of contiguous text objects}
```

*Implementation*

To accommodate the delete command, all text objects are given a *deleted* attribute. This is a flag which is initially unset. Applying a delete command to a text object, sets its deleted flag. When displaying the document in the Edit View, only those text objects, not marked as deleted, are displayed.

*Inverse*

The inverse of the delete command, unsets the deleted attribute of the text objects that were marked as deleted.

*Conflicts*

A conflict occurs under the following circumstances:

For delete,

- The list of text objects is not a continuous sequence of characters in the current version of the document.
- Any of the text objects in the list have already been marked for deletion.

For inverse delete,

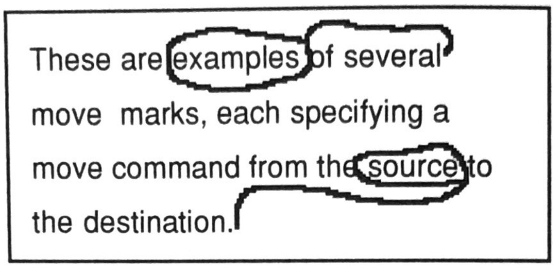
- In the current version of the document, the text objects surrounding the deleted text have been marked for deletion, caused by another delete command.
- The list of text objects is not a continuous sequence in the Edit View. This is to detect if other text has been inserted where the inverse delete would reinsert the deleted text. In order to make this test work properly, the insert and move commands actually insert text within the deleted text, thereby separating the deleted text. This special case is explained in more detail in the implementation sections of the Move and Insert commands.

### 4.3.2 Move

*Interface*

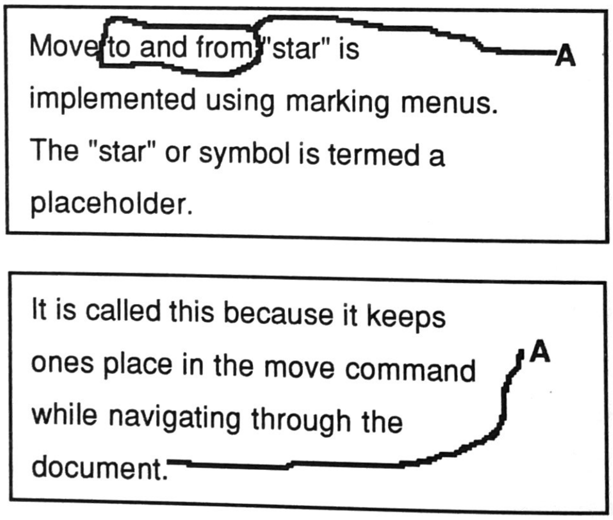
The regular move command is specified in a single mark. The source is circled and then a line is drawn to the destination as shown in Figure 4.8.

The broken move command is specified in two parts, *move to placeholder* and *move from placeholder*, Figure 4.9. Due to the limitations of the marking recognizer, both halves of



These are examples of several move marks, each specifying a move command from the source to the destination.

**Figure 4.8:** Move marks recognized by MATE
*A move mark is a single continuous line which first circles around the source text and then ends at the destination.*



Move to and from "star" is implemented using marking menus. The "star" or symbol is termed a placeholder.

It is called this because it keeps ones place in the move command while navigating through the document.

**Figure 4.9:** Broken move with move to placeholder and move from placeholder marks
*The broken move command is specified in two parts, each consisting of a single stroke. The first part, move to placeholder, is similar to a regular move command except that it ends in the margin and a selection from the placeholder marking menu. The second part, move from placeholder, is simply a line drawn from anywhere to the destination.*

the broken move command are specified by a single stroke. Also, only navigation actions are allowed between the two halves of the broken move command.

Move to placeholder starts exactly like a normal move command; the source text is circled and then a line is drawn. But instead of drawing the line to the destination, it is drawn to the margin. At this point, MATE recognizes the mark as a move to placeholder mark, and enters the placeholder marking menu mode. This marking menu mode is distinguished from the normal marking mode of MATE by inking a thinner line which is black in colour. The user ends the move to placeholder stroke by making a selection from the placeholder marking menu, Figure 4.10.

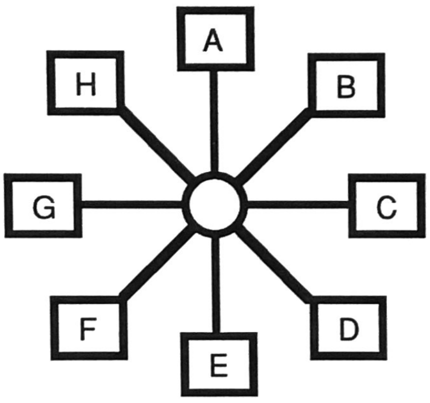


**Figure 4.10:** The placeholder marking menu
*The placeholder marking menu allows the user to choose from eight placeholder symbols, A - H.*

The broken move interaction ends with the move from placeholder mark. The move from placeholder mark is simply a line drawn from anywhere to the destination, because move from placeholder is the only possible action other than navigation. MATE shows the placeholder symbol specified in the move to placeholder stroke when the user starts to draw the move from placeholder mark.

*Representation*

The scope – source text – of the move command is represented by the text objects at each end of the scope, rather than a list of all the text objects within the scope, as is the case for the delete command. Specifying the scope in this manner allows the text objects within the scope to be deleted and moved, and allows text objects to be inserted within the scope. The destination is represented by the two text objects surrounding it. Therefore, the move command can be represented as:

```
move  {[source start text object] [source end text object]} to
      {[destination before text object] [destination after text object]}
```

*Implementation*

The set of text objects to move is determined from the start and end source text objects. These objects are then temporarily removed from the document and placed in a buffer. This allows the destination objects to be properly repositioned. The text objects in the buffer are then inserted between the destination objects, as shown in Figure 4.11.

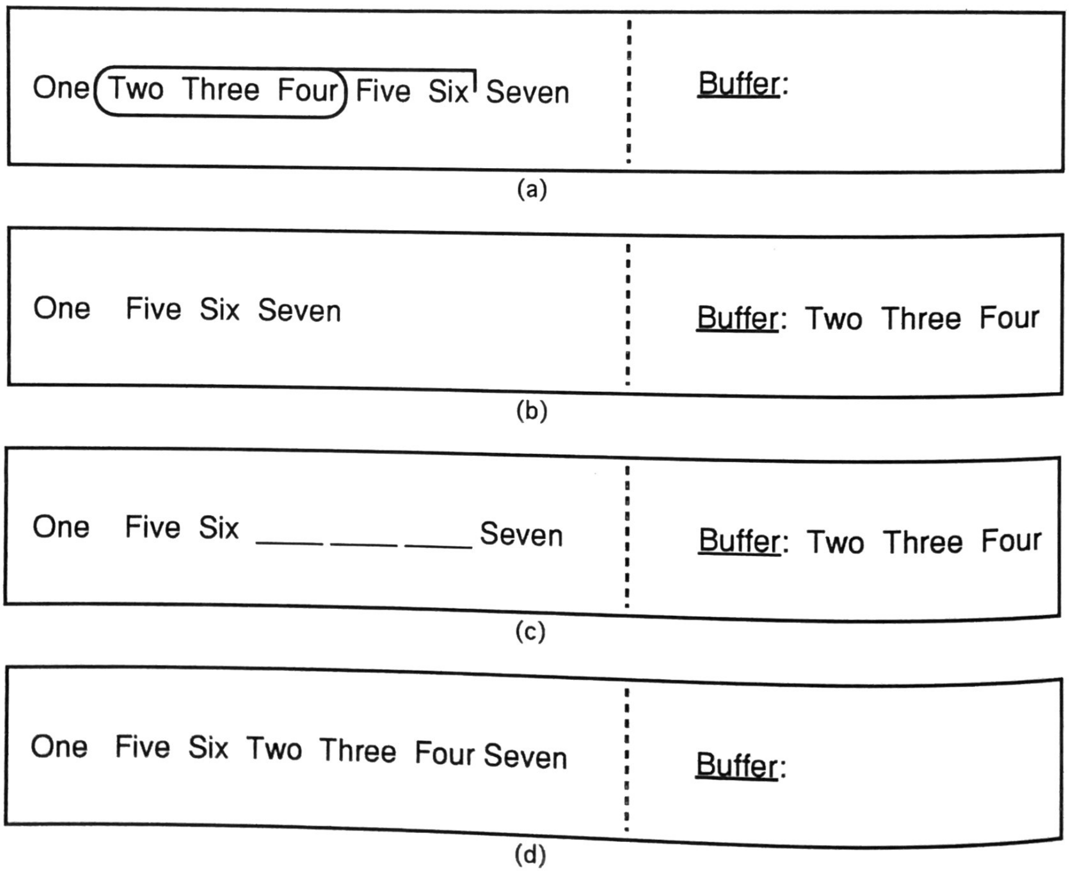


**Figure 4.11:** The implementation of the move command
*The move command shown in (a) is represented by the endpoints of the source – 2 and 4, and the destination – between 5 and 6. First, the source text is removed from the document and placed in a buffer (b). Then, null objects are inserted between the destination objects (c). Finally, the source text objects are moved from the buffer into the newly created text positions (d).*

*Inverse*

The inverse move can be thought of as a move command with the source and destination interchanged. However the inverse move command does not truly work in this manner. As the move command modifies the current location of the source text objects, the inverse move modifies the current location of these same text objects, but to a different destination. The key to making the inverse move work is in determining its destination.

A simple solution is to store, with the move command, the text objects immediately before the source start object and immediately after the source end object. In Figure 4.11, 1 and 5 would be stored as the inverse destination for the move command when it is saved in the command history list. Two observations are noteworthy about this method:

- We are really storing the context surrounding the source of the move command. Using contextual information is much better than positional information as the text may be moved by other move and insert commands.
- The method does not work in all cases, as the surrounding context may be lost. This is illustrated in Figure 4.12. The text objects originally surrounding "three" have been moved, thus destroying the context of the move "three" command. The inverse of this move command requires more information than storing the surrounding text objects.

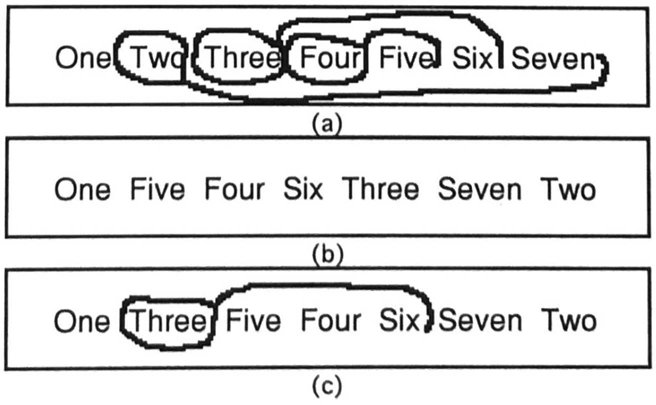

One Two Three Four Five Six Seven
(a)
One Five Four Six Three Seven Two
(b)
One Three Five Four Six Seven Two
(c)

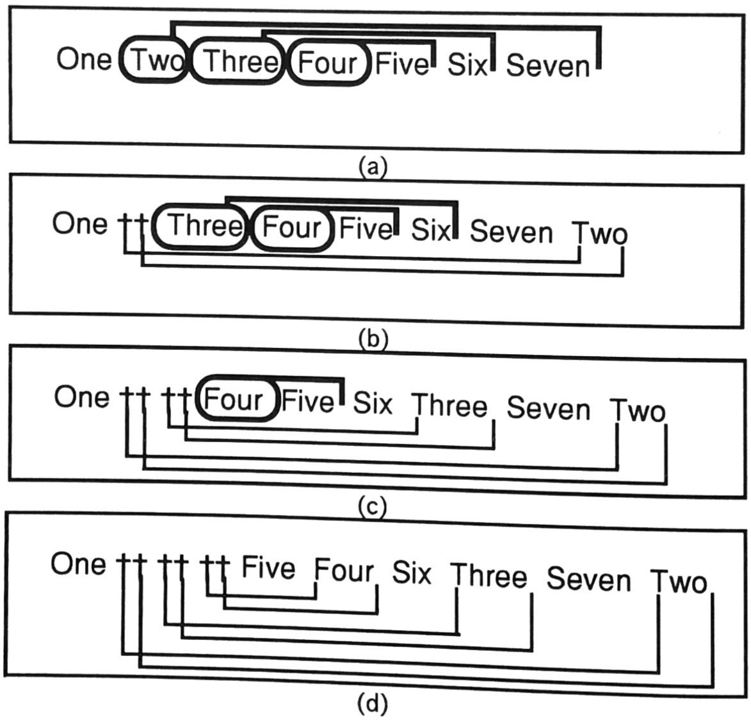
**Figure 4.12:** Loss of context after several move commands
*After all three move commands have been performed, the original context surrounding move "three" has been destroyed (b). Undoing move "three" should result in (c), but this requires more information than is provided by storing the original context of the command.*

The loss of context problem is solved by creating special pointer objects where the move command's source text used to be. These pointer objects are similar to character objects, in that they can be moved and deleted, but they are never displayed. These pointer objects are stored with the move command as the inverse destination. Figure 4.13 shows how pointers solve the loss of context problem for the example in Figure 4.12.

The inverse move then becomes

```
move  {[source start text object] [source end text object]} to
      {[pointer before text object] [pointer after text object]}
```

with the added action of removing the pointer objects.



One Two Three Four Five Six Seven
(a)
One †† Three Four Five Six Seven Two
(b)
One †† †† Four Five Six Three Seven Two
(c)
One †† †† †† Five Four Six Three Seven Two
(d)

**Figure 4.13:** The use of pointers in the move command to retain context
*By using pointer objects (†) in the move command, the destination for the inverse move command is clear. The three move commands can be done and undone in any order and always position the text in the proper places.*

## *Conflicts*

The following four of the move command's five conflict rules can be tested with the basic information available to the move command.

- A source boundary has been deleted: If either source boundary object has been deleted a conflict exists.
- The destination is within the source: If the set of text objects to be moved contains the destination objects, then a conflict exists.
- The destination has been deleted: If both destination objects have been deleted, there is a conflict.
- Text objects have already been placed in the destination: If the destination objects are separated, other objects have been inserted between them and therefore a conflict exists.

However, determining whether the source boundary objects have been moved independently of each other, requires additional information. This problem is illustrated in Figure 4.14. The problem is that the results of conflicting situations can be the same as the results of valid situations.

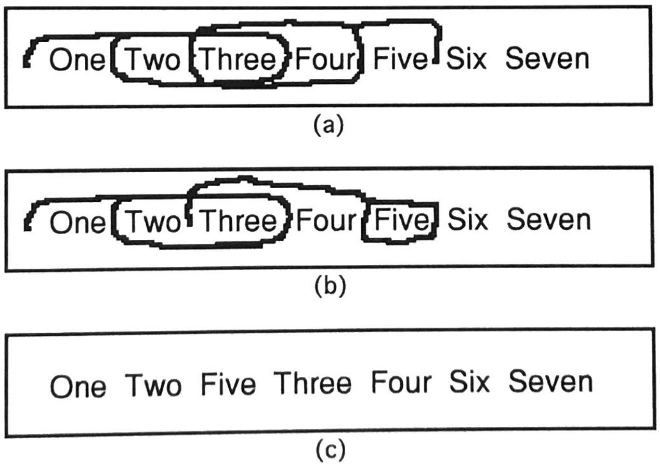

One Two Three Four Five Six Seven
(a)
One Two Three Four Five Six Seven
(b)
One Two Five Three Four Six Seven
(c)

**Figure 4.14:** Difficulty in determining independently moved source boundaries
*Applying move "three four" in (a) yields the same result (c) as move "five" in (b). If (c) is the result of move "three four" from (a), then move "two three" is invalid due to the conflict. However, if (c) is the result of move "five" from situation (b), then move "two three" is valid. More information is necessary to determine when move "two three" is valid (b), or invalid (a).*

The solution is to give each move command a unique Id, and to store these move Id's with each text object. To store this information, text objects are given the *move history list* attribute. The Id of every move command that acts on a text object is stored in the move history list of that object. To test whether the source boundary objects have been moved independently of each other, their move history lists are compared. If the histories do not match, a conflict exists. The inverse move command removes the move Id from the move history list of the text objects.

### 4.3.3 Insert

*Interface*

The insert mark is the caret symbol, as shown in Figure 4.15. To bypass the problems inherent in character recognition, the keyboard is used to specify the text to be inserted. First the user draws the insert caret, then a text entry window, that the user can type text into, is popped up (Figure 4.16). The insert command is completed either by pressing the Insert key to accept the command or by pressing the Escape key to cancel the command.

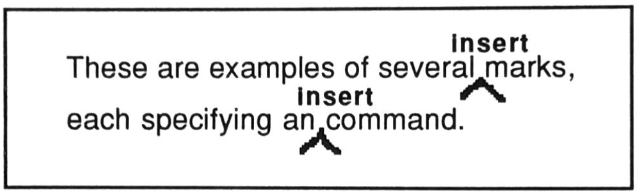

Insert
These are examples of several marks,
Insert
each specifying an command.

**Figure 4.15:** Insert marks recognized by MATE
*The insert command is specified by the insert caret and text entered via keyboard into a pop-up window.*

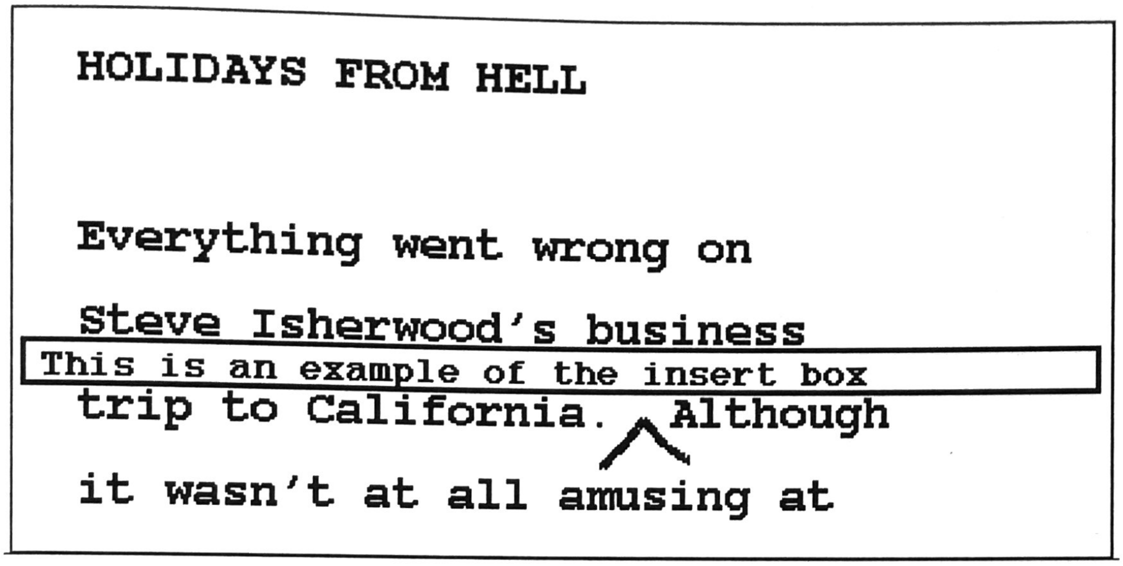
*Representation*

The insert command is represented as follows:

```
insert  {text string} at
        {[insert location before text object]  [insert location after text object]}
```

*Implementation*

The insert command first creates text objects for the characters to be inserted. These text objects are initially created in a text buffer. From this stage on, the insert command is treated as a move command, and would follow the method outlined in Figure 4.11(b), (c), & (d). The only difference in the implementation of the move and insert commands is that insert does not require pointer objects to be created.


HOLIDAYS FROM HELL

Everything went wrong on

Steve Isherwood's business

This is an example of the insert box

trip to California. Although

it wasn't at all amusing at

**Figure 4.16:** The pop-up insert text entry window
*After the insert caret has been recognized, MATE pops-up the insert text entry window. The user can then type in the text to insert. The insert command is completed either by pressing the Insert key to accept the command or by pressing the Escape key to cancel the command.*

*Inverse*

The inverse action of a command must leave the document in the same state as if the command was never performed. Therefore, the inverse insert command removes and destroys the text objects that were created by the insert command.

*Conflicts*

The insert command has the same rules of conflict as the destination of the move command.

- The insert location has been deleted: If both insert location objects have been deleted, there is a conflict.
- Text objects have already been placed in the insert location: If the insert location objects are separated, other objects have been inserted between them and therefore a conflict exists.

The inverse insert command has similar conflict rules as the delete command.

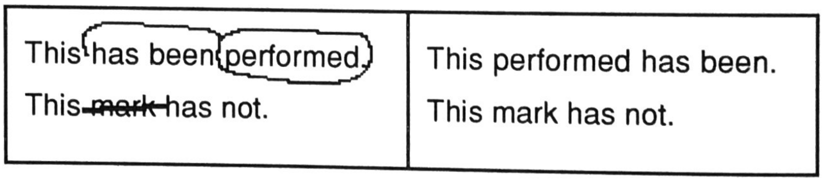
- The list of text objects is not a continuous sequence of characters in the current version of the document.

- A boundary object has been deleted: If either the insert's first text object or last text object has been deleted, a conflict exists.

## 4.4 Incorporation and Undo Support

An annotation is selected for incorporation and unincorporation by tapping on it with the stylus. This is part of a set of interactions, described in Section 4.2, that can be made on annotations. When a mark is selected, MATE first attempts to recognize it. If a valid interpretation can be made, and no conflicts are found, the command is incorporated (or unincorporated), and the current version of the document is updated. If a conflict exists, a pop-up message appears, stating that a conflict has occurred and the command was not executed.

Once an annotation is incorporated into the document, it is changed from a thick line to a thinner line, as illustrated in Figure 4.17. The feedback provided by line thickness, gives the user a means of visually identifying which annotations have been performed, and thus provides the user with a *graphical history mechanism.*



This has been performed.
This ~~mark~~ has not.

This performed has been.
This mark has not.

**Figure 4.17:** Line thickness as feedback for incorporated annotations
*All marks are initially thick lines.. Once an annotation has been incorporated, it becomes thinner. In the above example, the delete "mark" annotation has not been performed and is still a thick line. In contrast, the move "performed" annotation has been incorporated and is thin.*

Buttons are provided for the Undo and Redo last action mechanism, as shown in Figures 4.3 and 4.4. Undo last action can undo any number of editing commands back to the beginning of the editing session. This is accomplished via a history list which stores each command in a history entry. As all editing commands have an inverse, all that is required to be stored for each history list entry is the editing command's representation – the command and its parameters, which was described for each command in Section 4.3. As each entry is relatively small in size, the history list can be as long as a typical editing session.

Whereas Undo steps back through the history list, Redo steps forward up the last entry in the history list. The history list always ends at the last new command that was performed. Figure 4.18 illustrates how the Undo and Redo actions interact with the history list.

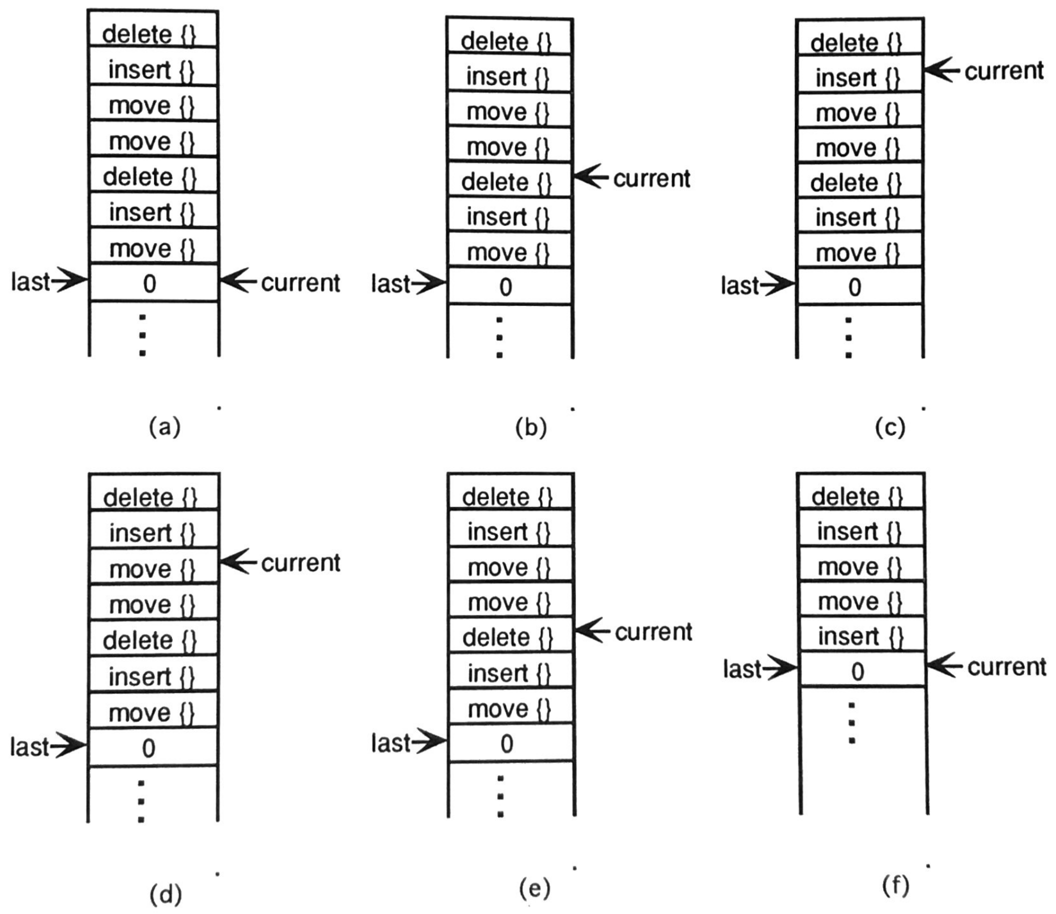


**Figure 4.18:** Undo / Redo last command and the history list
(a) shows the history list after seven editing commands have been performed. The last position pointer points to the end of the history list . The current location pointer separates the commands which have been performed – before it, and the commands which have been undone – after it. (b) and (c) show the effects on the current pointer as one command is undone (b), and when five more commands are undone (c). (d) and (e) show the effects on the current pointer as one command is redone (d), and when 2 more commands are redone (e). New editing commands are stored at the current pointer location. (f) show s the effects of entering an insert command from the situation in (e). Note that this changes the last position pointer so that any undone commands after the current entry cannot be redone.

## 4.5 Navigation

Navigation in MATE serves two main purposes – basic navigation, and relating the two versions of the document. Basic navigation includes moving the document up or down a page or several lines, with each of these commands able to act on both views of the document – *linked*, or on only one view – *unlinked*. Relating the two versions of the document is accomplished by the *Goto Text*, *Goto Annotation*, and *Goback* commands, which align the two versions of the document according to the text or annotation of interest.

Most of the navigation commands are accessed via *navigation mode*, which the user enters by pressing and holding the button on the pen down. The user interacts with this mode via a modified marking menu, shown in Figure 4.19. In addition to selecting menu items and making the corresponding marks, the user can make a *flick* mark to specify a page movement, and *L-shaped* marks to specify linked movement (Figure 4.20). Page flicks are vertical marks drawn quickly. They are distinguished from *page pushes* – move several lines – by the average speed in which the mark was drawn. Note that page flicks and linked moves are special marks and cannot be selected via menuing†.

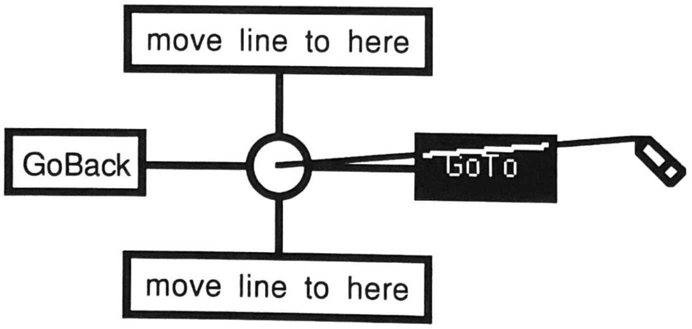
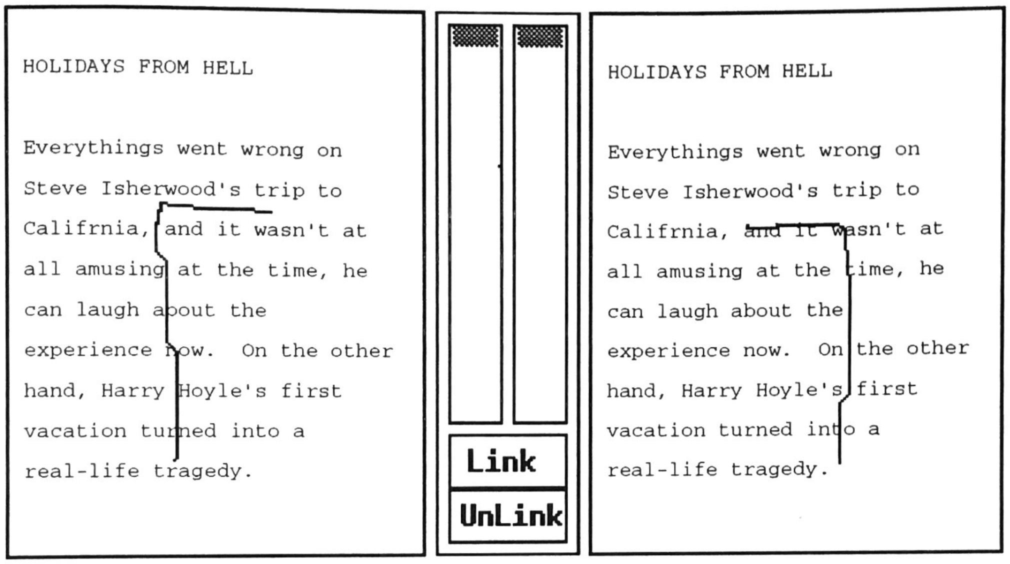


**Figure 4.19:** Navigation Marking Menu
*The navigation marking menu is invoked by pressing and holding the button on the pen down. The menu selections are move several lines – page push, Goto Text, and Goback. The navigation marking menu also has page flick marks and linked movement marks, which cannot be accessed once the menu is displayed.*

† Linked moves could be selected from a menu, as they can be thought of as hierarchic marking menu selections, which have recently been implemented (Kurtenbach, 1993). However, a page flick is distinguished from a page push by the speed in which the mark was made, and cannot be directly related to a menu entry.


HOLIDAYS FROM HELL

Everythings went wrong on
Steve Isherwood's trip to
Califrnia, and it wasn't at
all amusing at the time, he
can laugh about the
experience now. On the other
hand, Harry Hoyle's first
vacation turned into a
real-life tragedy.

HOLIDAYS FROM HELL

Everythings went wrong on
Steve Isherwood's trip to
Califrnia, and it wasn't at
all amusing at the time, he
can laugh about the
experience now. On the other
hand, Harry Hoyle's first
vacation turned into a
real-life tragedy.

**Figure 4.20:** Linked navigation marks
*The user starts by making the appropriate navigation mark – page flick or page push – and then makes a right angle to either the left or right.*

### 4.5.1 Basic Navigation

Navigation in mark-based systems is different than in mouse and keyboard based systems, in that a mark-based system does not require a text entry cursor – insertion pointer. The presence of a text entry cursor causes a problem with many text editing systems; there is no obvious cursor position after most navigation commands are performed. An examination of a small sample of editors reveals major differences in the placement of the cursor after navigation. For example, text 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 a mark-based system this problem does not exist.

There are several existing methods for navigating in a text document:

- Using a scroll bar to scroll or jump to a location
- *Pushing* the cursor into the edge of a window to display more text in that direction
- Making discrete jumps, usually of a page or half page in length, either by clicking in the scroll-bar area, or by commands entered with the keyboard

However, we decided to base MATE's navigation techniques upon the pen and paper analogy. The page flicks, mentioned above, correspond to turning the pages of a book bound at the top. To turn to the next page, the user flicks the page upward, and to turn to the previous page, the user flicks the page downward. Moving several lines corresponds to pushing a continuous sheet of paper. To move forward in the document, the user pushes the document upward, to move backward in the document, the user pulls the document downward.

Note that MATE's basic navigation is based on moving the document, rather than moving a window over the document. Moving the document has the advantage that it makes sense to specify the navigation commands in the document area rather than the scroll-bar. As the document area is much larger than a scroll-bar, the user can specify navigation commands in the document area much easier and faster than with a scroll-bar (MacKenzie, 1992). However, moving the window using the scroll-bar has the advantage that the user can jump directly to a specific location in the document, as opposed to making a series of page movements. As the two methods are not mutually exclusive, moving the window via the scrollbar was also implemented.

*Linked Navigation*

As mentioned in Section 3.4.6, all linked navigation is relative movement dependent. This means that both views move the same amount regardless of their content. To change a basic navigation command into a linked navigation command, the user starts by making the appropriate mark and then continues the mark at a right angle to either the left or right, as shown in Figure 4.20†.

### 4.5.2 Relating The Two Views

The solutions of highlighting text and aligning the two views, mentioned in Section 3.4.6, are integrated into both the Goto Text command and the Goto Annotation command. Goto Text is applied to a word in the document, in either view, via the navigation marking menu (Figure 4.19). Goto Annotation is applied to an interpretable annotation in the Annotation View, via the annotation marking menu (Figure 4.5).

† Note, in an earlier version of MATE navigation was linked or unlinked via a mode switch, as shown in Figure 4.4. This was disabled in the current version of MATE, in order to test the interaction described in this section.

### *Goto Text*

The Goto Text command is applied to a word in either the Annotation or Edit View – *master view*. The other view – *slave view* – is then modified in the following manner:

- If the word is not visible in the slave view, the slave view is adjusted so that the word appears on the same line in both views. The word is also highlighted in the slave view.
- If the word is visible in the slave view, then the word is highlighted in the slave view, but the position of the document in the slave view is not adjusted, except in the following case.
- If a second Goto Text command is applied to the same word, then the word remains highlighted, and the slave view is adjusted so that the two views are aligned.

### *Goto Annotation*

Whereas the Goto Text command is applied to a word, the Goto Annotation command is applied to an annotation. The selected annotation is first interpreted into an editing command. Goto Annotation is then applied to the parameters of the editing command in a similar manner as the Goto Text command, with the following exceptions:

- If all the parameters cannot be displayed at once, the Edit View is adjusted so that the first parameter is aligned in both views. Subsequent invocations of Goto Annotation will align the views so that each parameter in turn is aligned in both views.
- If the parameter is not visible in the Edit View, the Edit View is adjusted so that the beginning of the parameter is aligned in both views. All of the text objects are highlighted in the Edit View.
- If the parameter is visible in the Edit View, the parameter is highlighted in the Edit View, but the position of the document in the Edit View is not adjusted. Note that subsequent invocations of Goto Annotation are used to step through the parameters of a command.

### *Goback*

In many cases the user may use the navigation and Goto commands to examine a section of the document, with the intention of returning to a previous point of interest. This is especially true for a move command which uses placeholders, for which the source and the destination are usually far apart. The Goback command is the navigation mode's Undo Last command. To accomplish this, a *navigation history list* was implemented.

This history list stores points of interest to the user. When the Goback command is executed the Annotation and Edit Views are adjusted to the previous entry in the list.

The main problem in building the navigation history list is in determining what constitutes a *point of interest*, and therefore when should a document position be stored as an entry in the navigation history list. In contrast with the command history list, the entries in the navigation history list are not explicitly specified. The following criteria are used in deciding when to store a document position in the navigation history list:

- When an annotation has been incorporated, or if an editing command has been executed.
- When a Goto Text or Goto Annotation navigation command is to be executed.

If executing editing commands and incorporating annotations are thought of as implicit Goto commands, Goback is the inverse of Goto.

## 4.6 Summary

The interfaces of MATE's three modes are similar; the document is shown in the top portion of the window, and utility functions are placed in the bottom portion. Our main focus is on the document area, in which the editing commands are specified and the annotations are made.

Several annotation support functions are provided via the annotation marking menu: Erase, Goto Annotation, Hide, and Goback. To support annotations from several sources, MATE provides functions which are accessed through buttons in the marking area of the MATE window. These functions are: Load, Save, Hide, Show, and UnHide.

After making all the additions needed to support the editing commands, each text object has the following attributes:

- An *ASCII character value*
- The *original position in the document*
- The *original position in the Annotation View*
- The *current position in the document*
- The *current position in the Edit View*
- The *deleted* flag

- The *pointer object* indicator. Specifies whether an object is a regular text object, a pointer before object, or a pointer after object.
- The *move history list*. A list containing the Id's of the move commands that have acted on the text object.

Three editing commands have been implemented, delete, move, and insert. The markings used to specify these commands are all single strokes, except for the broken move. All commands have an inverse, and all commands only modify the attributes of the text objects, except for insert which creates text objects.

The delete mark is a horizontal stroke. Text is marked for deletion by setting the deleted flag. Inverse delete unsets the deleted flag.

The regular move mark consists of a circle around the source text, followed by a line ending at the destination. Broken move consists of the move to placeholder mark, and the move from placeholder mark. Move to placeholder is similar to a regular move mark, except that it ends in the margin where a placeholder symbol is chosen from the placeholder marking menu; move from placeholder is just a line drawn to the destination.

The move command changes the current position attributes of the modified text. To support the inverse move command, pointer objects were implemented to solve the problem of loss of context. To properly identify conflicting move commands, each move command is given a unique Id, which is stored in the command history list as well as in the move history list attribute of the source text objects.

To specify an insert command or annotation, the user first draws the caret symbol. A text entry window, into which the user can type, then appears. The insert command or annotation is completed by pressing the Insert key to accept the command or by pressing the Escape key to cancel the command. The insert command first creates text objects in a text buffer, and then moves these text objects to the insert location. Inverse insert destroys the created text objects.

An annotation is selected for incorporation and unincorporation by tapping on it with the stylus. After an annotation is incorporated into the document, it is changed from a thick line to a narrow line. The command's interpretation and the attributes of the text objects contain all the information necessary to either Do or Undo the command.

The Undo Last and Redo Last functions are accessed via buttons in the utility area of the window. These functions step, backward and forward respectively, through the command history list. The current state of the document, and the information stored with each command in the history list are sufficient to undo or redo each command in order.

Navigation is accomplished via navigation mode, and the navigation marking menu. The user enters this mode by pressing and holding the button on the pen down. The basic navigation commands borrowed from the analogy of flicking and pushing / pulling a page of a book. Normally these commands act on only one view of the document – unlinked, but by continuing the flick or push mark at a right angle to either the left or right, both views are affected by the navigation command – linked.

Three special navigation functions were implemented to aid the user in relating the two versions of the document: Goto Text, Goto Annotation, and Goback. Goto Text is applied to a word in either view – the master view, and adjusts the other view – slave view – so that the word is aligned in both views, and highlighted in the slave view. Goto Annotation is similar except that it is applied to an annotation in the Annotation View, and it adjusts the Edit View so that the parameters of the command are aligned and highlighted. Goback steps back through the navigation history list, and is the inverse of Goto.

Relating back to our research question, the design and implementation of MATE show that a system which integrates the two uses of marks, as annotations and for specifying editing commands, can indeed be built. In addition to the benefits mentioned in earlier chapters, MATE's implementation showed the following

- The Do and Undo by selection can be achieved by a tap of the pen, and in the process provides a graphical history mechanism.
- Undo Last and Redo can be implemented via the command history list.
- By borrowing from the pen and paper analogy, navigation can be integrated into the system.

Chapter 5

# User Testing

Although MATE can be used in several stages of the collaborative writing process, user testing all of these is beyond the scope of this thesis. Instead we concentrate on the Incorporation Mode of MATE, as this is the central part of our research.

In general, user testing may examine both the usability and usefulness of a system. However, testing the usefulness of MATE as a collaborative writing tool would require an extensive study. Therefore we mainly examine the usability of MATE with the following goals in mind:

- to obtain information so that MATE can be made more usable,
- to determine how usable MATE is.
- to gain some insights in to the usability and usefulness of the general design concepts when possible.

A comparison with existing methods for incorporating annotations into a document was not carried out. There are several reasons for this:

- MATE is a prototype, which is missing a lot of the functionality of a complete system. A test with an existing method would mainly point out the missing functionality of MATE, which is not a primary concern in this thesis.
- The main existing method is to print out a document and send it out to the collaborators, marking up the printed document and sending it back, and incorporating the changes by editing the document. A comparison with this method would have been premature as it would not yield useful information about the usability of MATE in its current state.

## 5.1 Design of the Usability Study

The usability study is designed as an experiment consisting of the following four parts:

- A *training session* in which the user was trained how to use MATE. The purpose of the training session is to introduce the main concepts and to give the subject practice in using MATE's interaction techniques.
- A *task session* in which the user performed an editing task using MATE. Ideally, the testing task should be designed as a task which may exist in a real-life application, however, as MATE is a prototype system, the task was designed so that only the existing features of MATE would be needed to perform the task.
- A *questionnaire* which the user answered after the task session was completed. The questionnaire is designed to help determine the users' understanding of the concepts, and to provide a comparison to the observations during the training and task sessions.
- An *interview*. This covered more general questions than the questionnaire, and allowed the user to add any comments that the questionnaire may have missed.

The data collected from the experiment consisted of:

- logs of the training and task sessions,
- videos of both the subject and the computer screen,
- the questionnaire, and
- the interview.

*Training Session*

The user training is not meant to test how easy it is to learn how to use MATE. As this would involve much design work on user training which is not part of this thesis. Therefore the subjects were first guided through a training session to ensure that they obtained at least a basic understanding of MATE and the underlying concepts behind MATE.

The training session was guided by a training manual and a training document, shown in Appendix A. The subjects were asked to perform the tasks given in the training manual. They were allowed to ask questions and receive additional guidance by the experimenter on how to use the system. The experimenter also ensured that they actually performed the tasks.

*Task Session*

In order to be useful for testing the incorporation mode, the editing task had the following requirements:

- the document should be in an almost final state.
- most of the annotations should be small explicit corrections rather than more general annotations such as "reword".

It would be difficult to design a task that could test all of the functionally of MATE and also not be affected by the limited functionality of MATE. Therefore the task was designed to test the following:

- the navigation features of MATE,
- the user's understanding of the relationship between the two windows,
- the usability of the editing commands, i.e. the users' ability to articulate the mark and the users' ability to remember the mark.

In contrast to the training session, the subjects were instructed to improve the document as they saw fit. They were not given guidance, except in specific cases which are noted in the results. The annotations given were to be thought of as suggestions only. The task document is shown in Appendix A.5.

*Questionnaire*

The questionnaire examined the subjects' understanding of the editing commands, incorporation, navigation, and the usability of the system. The Questionnaire is shown in Appendix A.6.

Interview

The interview was designed to find information which the questionnaire missed. The standard interview questions are shown in Appendix A.7.

## 5.2 Results and Recommendations

Observations from the experiment are noted in Appendix B. B.1 contains observations from the training sessions, B.2 from the task sessions, B.3 summarizes the answers to the questionnaire, and B.4 contains a condensed transcription of the interviews. In this section the results from the experiment are organized by whether they pertain to MATE in general, to editing and incorporating annotations, or to navigation.

## General

*Slow / erratic response time*

At some points in the sessions the system took two minutes before responding to a navigation command. The exact reason for this is unknown, but may be attributed to swapping pages of memory or network traffic. Also the asynchronous nature of the X window system caused the system response time to be irregular. Such response time causes many problems and must be corrected to make MATE a more usable system.

*Amount of text displayed*

The font size used in the usability study was large – 18 point. This was to make it easy for novice users to make markings. However, this meant that only 18 characters per line and 12 lines of text were displayed at one time. Several subjects commented that they would have liked to see more text at a given time. More testing is required to determine the best font size for general use, but it is clear that the 18 point font is too large.

## Editing

*Command Set*

For the most part, subjects found the command set adequate for the given task, but a couple subjects requested more functionality especially in the form of *replace* and *modify* commands. Replacing text is functionally equivalent to deleting and inserting text, but combines both in a single command. Modifying text transfers the text editing from the main text editing area to the pop-up text entry area, with the intention of facilitating many small edits on a piece of text.

The delete, replace, and modify commands are related in the following manner:

- Delete is a subset of replace without any text inserted.
- Replace is a subset of modify in which all the existing text is deleted before any text is entered.

There are several ways of adding the modify and replace commands to the command marking set. One is to use the circling mark found in the beginning of the move command to select the text of interest. Another is to have the two existing delete marks – horizontal line to the left and horizontal line to the right – mean two different commands.

One solution in which both commands are added is as follows:

- Modify is specified by circling text,
- Replace by a horizontal line to the right (or left),
- Delete by a horizontal line to the left (or right).

*Undo / Redo Last*

Undo last was used very infrequently. Only one subject used it several times in a row, followed by several redos. There were several occasions in which undo last would have been useful for the subject, but the subject did not think of using it.

*Delete*

In general users had little difficulty with the delete command. However, there were a couple of problems: deleting single characters and deleting more than one line of text.

To delete a single character the user must make a very short horizontal mark. This is a problem as MATE has a lower limit on the length of recognizable marks in order to remove accidental and spurious marks. The length of the horizontal line must be drawn within a very small tolerance. One solution would be to add the slash mark to the marking set for deleting single characters.

To delete several lines, the user is forced to delete every single line, one at a time. This might be fine for a few lines, but is unacceptable for many lines. One solution can be taken from our observations of the everyday use of markings. Often when a paragraph or page is to be deleted, the annotator draws a large slash mark from corner to corner.

*Insert*

The recognition rate for the insert mark was very low. Even worse, it was sometimes misinterpreted as a delete command. Even when recognized, there were cases in which the insertion point was off by a character. The recognition rate of insert must be improved, and the system's distinction between insert and delete must be brought more in agreement with the user's expectations.

The insert box can be set to any size – number of lines of text – via a preferences file. For the usability study it was set to one line. There were several instances in which a subject wanted to enter several lines of text. After typing past the width of a line of text, the user could not see the characters being typed. To see these characters, the subject typed a new-line character. The subject could then at least see what was currently being typed, although all the previous lines could not be seen. To solve this problem the insert box should be resizeable and scrollable. The text should also word wrap.

We assumed that most of the editing would take place at the word level, as opposed to the character level. To accommodate this, MATE would automatically prepend and append spaces to the inserted text. This caused problems when a subject wanted to modify words, especially when adding suffixes. The subject noted the conflicting issues and would have preferred to manually enter spaces in order to have the control over when spaces were added. One solution would be to have a *word mode*, which could be toggled between automatically adding spaces and not adding spaces. But such a subtle mode is likely to be forgotten. Another solution would be to use one insert character – ^ – for insert word with spaces, and the other – v – for insert characters without added spaces.

*Move and Broken Move*

Subjects had little difficulty in specifying both move and broken move commands. A few bugs in the system were noticed, but the subjects had no problems with the interaction methods.

### Annotating

An important point, noted by one subject, is that a different action – annotations menu – occurs when pointing to a mark versus not pointing to a mark – drawing. Although, these actions are generally what is wanted from the system, an annotation mark cannot be made in close proximity to an existing mark, for it will be interpreted as command on the annotation. This makes some annotations, such as a handwritten note in the margin, impossible to draw. Some method is needed to allow marks to be made on top of other marks.

### Navigation

There were several problems with MATE's navigation. The major problem was the button on the pen. Several users found holding the button down while drawing a navigation command to be physically difficult. The button actually broke during Subject 1's training session. Also there was a slight delay in the recording of the pen movement when the button is first pressed down. It is unclear whether this is a hardware or software problem.

An interesting note is that Subject 1 had to use the mouse for entering navigation commands as the button on the pen was broken. This allows us to informally separate the interaction mechanisms from the input device being used. It was clear from our observations that the mouse and mouse buttons were tracked much more accurately than the pen. This made a large difference in that the subject had little or no difficulty with the mouse in specifying the navigation commands, including page flicks and linked navigation.

The combination of page flicks and linked navigation with a marking menu pushes the limits of the interaction technique. This technique requires almost expert usage from the beginning and does not lend itself well to incremental learning, both cognitively and physically. As one subject put it "There may be a limit to how many commands you can squeeze into a pen motion!" Although observations show that subjects can use the interaction technique, it would be useful to design alternatives to compare with.

All of the subjects were used to the idea of moving the window over a document via a scroll-bar. In MATE, the document is moved and the window is still. This caused some initial confusion as to the direction of page flicks and pushes, but the subjects quickly learned the correct directions. Another important point is that flicks and pushes are relative movements, in order to navigate to a specific location other mechanisms must be provided. In MATE, these mechanisms were provided by scroll-bars.

### *Navigation Mode*

The main problem with navigation mode, aside from using the button on the pen, was knowing when one was in this mode. The only visual feedback provided was that a thin black mark was drawn. A solution is to show a different cursor when in navigation mode.

### *Page Pushes*

Subjects noted that the page push happened too quickly for them to know how the document actually moved. A much better method, closer to the page push analogy, is to *grab* the document with the cursor, and move the document as the cursor is being moved, rather than after. Visual feedback, such as displaying a hand-shaped cursor, would also enhance the interaction as well as the analogy.

*Page Flicks*

Page flicks were very sensitive to any system delays. Also the user had no feedback whether a navigation mark would be interpreted as a flick or a push.

*Linked Navigation*

Conceptually, the subjects had little difficulty with linked navigation. However, there was a lot of difficulty physically. One subject resorted to using the mouse for linked navigation and achieved much better results. It seems that the problems with linked navigation are with the implementation of the pen rather than the technique itself.

*Goto*

The subjects seemed to understand both Goto Annotation and Goto Word. However, there were some conceptual difficulties. Only one user answered that Goto Word was useful in lining up the windows – the primary function of Goto Word. The labels for both commands was the same – Goto – but in different marking menus. This caused some confusion and should be changed.

One subject used the Goto Annotation command twice to see the two parameters of a broken move command. When MATE jumped to the destination and highlighted it, the subject noticed that part of the source text was visible. But when the user pushed the page to see more source text, the highlighting disappeared, and the user lost track of where she was in the document. This brings about the issue of how Goto, specifically highlighting, should affect the document and how it should interact with the other navigation commands.

*Goback*

Goback was used once among all subjects. Although this lack of use might have been due to a lack of need, it also seemed apparent that the subjects were not sure what it actually did. Further study is needed to determine if a function such as Goback is useful, and how such a function should be designed and implemented.

## 5.3 Summary

A usability study of MATE in Incorporation Mode was carried out. This study was in the form of an experiment consisting of a training session, a task session, a questionnaire, and an interview. Two pilot subjects and 3 subjects participated in the experiment. In addition to the questionnaire and interview, data was also obtained from logs of the training and task sessions, and videos of both the subject and the computer screen.

MATE was found to be usable, but could definitely use improvements. The following observations and recommendations can provide some guidance on how to make some of these improvements:

- *Slow / erratic response time:* The response time was unacceptably slow at times.
- *Increase the amount of text displayed:* The font size should be made smaller – than 18 point – to accommodate more text on the screen at one time.
- *Add Replace and Modify commands to the command set:* Although the existing command set was sufficient for the most part, replace and modify commands would definitely be useful.
- *Undo / Redo Last used infrequently:* Subjects either did not really have a need, or else were not used to having the undo / redo capability.
- *Minor problems with Delete command:* Subjects had little difficulty with the delete command, except when trying to delete a single character, and when deleting several lines. A slash mark could be used to solve both the single character problem, and the multiple line problem.
- *Major problems with Insert command:* The recognition rate for insert was unacceptably low, and was often misinterpreted as a delete command. The insert text entry box should be resizeable, scrollable, and have word wrapping. Some method of distinguishing between inserting characters and inserting words would be helpful. One solution is to use one insert character – ^ – for insert word with spaces, and the other – v – for insert characters without spaces.
- Move and Broken Move had no real problems.
- *Actions on annotations versus drawing on annotations:* A method for allowing marks to be made on other marks is necessary.
- *General navigation problems:* The button on the pen was very unreliable, difficult to use, and caused several problems. Observations of a subject using a mouse show that the interaction techniques are valid, and that the implementation of the pen needs to be improved. The combination of page flicks and linked navigation with a marking menu stresses the interaction technique, requiring almost expert usage from the beginning. Alternative designs should be considered.

- Visual feedback for navigation mode should be provided.
- *Modify page push interaction:* The page push interaction should be changed to a document grab interaction, in which the document moves as the cursor is being moved.
- *Problems with page flicks and linked navigation:* Page flicks and linked navigation were very sensitive to any system delays. This and the general problems with the pen and navigation must be solved in order to make page flicks and linked navigation more usable.
- Labels for Goto Word and Goto Annotation should be distinct.
- Minor modifications to Goto Annotation should be investigated.
- Further study is needed to determine if Goback is useful and if so, how it should be designed and implemented.

Chapter 6

# Conclusions and Future Work

This thesis began with the observation that marks have been used to specify commands in text editors, and that people use marks to annotate text documents, but that these two uses have never been integrated. We noted that this integration was related to three areas of research: asynchronous collaborative writing, the visibility of markings, and interaction languages which are understood by both people and machines. To explore this integration, MATE was designed, implemented, and user tested.

The three research areas are interrelated and thus difficult to separate in the earlier chapters of this thesis. In this chapter, we relate the knowledge we have gained to the three research areas, and how each area has affected the others.

## 6.1 Conclusions

*Visibility of Markings*

The visibility of markings allowed the commands specified by markings to be placed in the context of the text document. This allowed the graphical history mechanism, described in Section 4.4, which enables the user to determine which commands have been incorporated and which have not.

The visibility property also allowed direct interactions with the markings, thus allowing actions to be applied to marking commands in an easy manner. Such actions include, selection for Doing or Undoing, and marking menu interactions to Hide, Erase, Goto, or Goback.

Visibility is by no means restricted to markings; a mouse and keyboard system could also be designed to have visible commands and annotations. A speech-based system would need visual information, such as graphical icons associated with an utterance of speech. But this would still be lacking, because most of the information – the speech – is still not visible.

*Common Interaction Language*

The issues and benefits of an interaction language common to both people and computers were mentioned in Section 2.4. The problems with a common interaction language are the constraints placed upon it by the capabilities of both people and computers. In the case of markings, the computer can rely on timing and ordering information, whereas a person can only rely on visual information. Also people have already invested many years in learning how to read and write certain marks such as the alphabet. Therefore, it is more compelling to make the computer understand the marks people use, rather than find a solution closer to the abilities of the computer.

There are many benefits of a common interaction language. In the context of our asynchronous writing scenario, these benefits include enhancing the annotation reader's understanding, enhancing the annotation writer's understanding, reducing the effort needed to convert the annotation into an editing command, and enabling "What if?" scenarios.

An important point to note is that, in our system, the computer does not need to understand all of the marks that people understand. In the worst case, the system would not understand any of the annotations and the user would have to enter all of the commands manually. So our worst case reduces to what is currently being done. Each annotation that is understood by the system adds to its benefits, whereas annotations which are not understood do not detract any benefits.

*Asynchronous Collaborative Writing*

The integration of annotations and editing commands provides several benefits for asynchronous collaborative writing. These benefits include the graphical history mechanism, enhancing users' understanding of the editing marks, and reducing the need to convert annotations into editing commands.

We have developed a text editing system which supports several aspects of the asynchronous collaborative writing process. Despite some of the problems with the implementation, MATE has proven to be usable. The usefulness of MATE as a system for asynchronous collaborative writing was not determined, and requires further study once some of the implementation issues are dealt with.

*General*

Several insights were gained by applying observations in one research area to another. Some of these are the following:

- The key to filling the gap in the asynchronous collaborative writing process is the use of a common language among people and computers.
- A common interaction language was developed from proofreaders' marks found in the marking up of text documents.
- The visibility of markings provided the mechanism for integrating the two uses of markings – as annotations and for specifying editing commands – in a seamless manner.

## 6.2 Future Work

Several details and problems in the implementation of MATE impeded our research. Therefore future work is divided into two parts: implementation improvements to obtain more useful research results, and future research and design work.

### 6.2.1 Implementation Improvements

Several recommendations for future work resulted from the user testing. To recap:

- *Increase the amount of text displayed:* This can be accomplished by reducing the font size.
- *Add replace and modify commands:* Based on the results from the user testing, replace and modify commands, described in Chapter 5, would be the most useful commands to add to MATE's existing command set.
- *Delete:* To overcome difficulties in deleting single characters and large phrases, use a slash mark. A small slash over a single character would delete it, and a large slash over several lines would delete those lines.
- *Insert:* First, the recognition rate needs to be improved. Also a method of distinguishing between inserting words and inserting characters would be helpful, and the text insertion box should be resizable, scrollable, and have word wrapping. This text insertion box could actually be a special version of MATE in Edit Mode.

*Data Storage*

Currently there is one text object for every character. As each text object requires approximately 50 bytes for all its information, large amounts of memory are required for relatively small text files. This may be the cause of the delays noticed in the user testing. A better implementation would group characters together into a single text object, thus reducing the storage overhead.

*Navigation*

Users experienced physical difficulty with navigation due to the need to push the button on the pen down while specifying a navigation command. Erratic system response time and a lack of feedback also contributed to the difficulty in navigating. Solving these problems should eliminate most of this difficulty.

*Actions on Annotations*

There is a problem in the current implementation in that marks cannot be drawn on existing marks, because a penning down on an existing mark automatically sends MATE into annotation marking menu mode. A solution which allows marks to be made on top of other marks is needed, such as some method of temporarily turning off the annotation marking menu.

Another problem is that erasing or hiding marks is achieved by selecting each mark and then making the appropriate marking menu selection. This is fine if only a few markings are to be erased / hidden, but becomes cumbersome if a group of markings are to be erased. One solution is to have an eraser mode which would then erase any marks which the eraser comes in contact with. Another solution is to have a scoping mechanism, such as circling, to select a group of marks on which an action such as erase could be performed.

### 6.2.2 Future Research and Design

*Speech*

As mentioned in Section 2.1.3, spoken annotations would be very useful, even if they are not interpreted into commands. The speech annotations could be displayed as icons, and be associated with a section of text. Another extension would be to coordinate a speech annotation with a marking annotation so that an animation of a marking annotation could occur while a spoken annotation is being played.

*Move*

Although the subjects in the user testing had little difficulty with the move command, there are several improvements which could be made. Circling large sections of text becomes very difficult, and is impossible if the section is larger than the window size. One solution is to use brackets to specify the end-points of the section of text.

*Insert*

The other main extension for insert is to use the pen with character recognition instead of the keyboard. The keyboard was initially used because a character recognizer was unavailable at the time of implementation. Allowing both pen and keyboard input would be useful as the keyboard is a much better input method for entering large amounts of text.

*Broken Move and Placeholders*

Several design alternatives were generated for the broken move command and how to specify placeholders (Section 3.4.2). Only one of these was implemented – using a standard set of symbols to specify the placeholder, and a forced pending move. The alternatives to specifying placeholders are: drawing the placeholder symbol for both the source and destination, and drawing the placeholder symbol for the source only, leaving the symbol visible for the destination command. The alternatives for the broken move interaction are: the unforced pending broken move, and to treat a broken move as two distinct move commands – move to placeholder and move from placeholder. The two distinct move commands alternative allows multiple clipboards, which could be treated as editable text documents. This and the other alternatives show promise and should be examined further.

*Conflicts*

Some conflicts exist because the user's intention requires two commands to be executed simultaneously, which causes an order dependency conflict (Section 3.4.5). For example, the intention to transpose two phrases requires two move commands to be executed at the same time. To solve these conflicts, the integrated commands should be added to the command set. The three compound commands identified so far are transpose, replace, and replace by move.

*Navigation*

An alternative solution the our existing navigation technique is to use a touch tablet in the other hand to specify navigation commands. The paper analogy is even stronger as brushing one's finger against a touch tablet is very similar to turning the pages of a book. Another possibility is to use the touch tablet in combination with the pen. For example, the touch tablet could be used to enter navigation mode, and distinguish between page pushes and page flicks, and between linked and unlinked navigation.

*Documents containing text and marks*

The current version of MATE treats the marks and text differently. In some cases users may want marks to remain as part of the text document, either within a single editing session or across several editing sessions. An example might be a reminder to look up a reference, or a note to oneself to reread a section. Such a situation occurred in the user testing when a subject wanted to make a note to himself by drawing in the Edit window, but was unable to as the marks were immediately interpreted.

*Single View version*

The decision to use two views, made in Section 3.3, was a central decision in determining the research directions of this thesis. The single view alternative would shift much of the research towards how the system would automatically relate the two versions of the document. As mentioned in Section 3.3, it was important to examine the two view approach first as the automatic relating of the single view approach may not have solutions in all cases. The best solution is probably a hybrid in which the single view is the default with two views available in those cases that cannot be handled properly in a single view.

*User Testing*

Many questions were not asked or answered by the user testing covered in this thesis. These mainly concern features and functions which are not used often, such as Undo / Redo last, and Goback. Also to fully test the functions intended to help users relate the two views, .large documents must be used and the original and current versions must differ by a large amount to strain the user's understanding of the relationship between the two versions.

## 6.3 Discussion

This thesis could be considered more as exploratory research than as a solution to a particular problem. We began with the observation that a gap existed in the use of markings and in the computer support of asynchronous collaborative writing. Instead of concentrating on one particular problem, we focused on one particular solution – the design, implementation, and user testing of MATE. By doing so, we have laid the basis for future research in several areas:

- MATE could be developed further, into a system that supports the asynchronous collaborative writing process better.
- The visibility of markings, and other properties of markings, could be explored further by analyzing which properties are used in applications and which have yet to be exploited.
- Common interface languages can be examined and developed more. The insights gained in our work with markings might be applicable to other types of common interface languages such Natural Language Understanding. Also a more extensive marking language understandable by people and computers would be beneficial.

Our research has shown that each of the above areas of research has a lot of potential, and is worthy of further investigation.

# References

Buxton, W. (1990). The "Natural" language of interaction: A perspective on nonverbal dialogs. In *The Art of Human Computer Interface Design.* Reading, Massachusetts: Addison Wesley.

Buxton, W. (1991). *The Pragmatics of Haptic Input.* in press. Also The Pragmatics of Haptic Input,*Tutorial Notes, CHI' 90*, Seattle, Washington.

Buxton, W., Sniderman, R., Reeves, W., Patel, S. and Baecker, R. (1979). The evolution of the SSSP score editing tools. *Computer Music Journal, 3*(4), pp. 14-25.

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

Carr, R. M. (1991). The point of the pen. *Byte*, February, pp. 211-226.

Chow, D. and Kim, J. (1989). Paper Like Interface for Educational Applications. *Proceedings of National Educational Computing Conference '89,* June 20-22, Boston, Massachusetts, pp. 337-344.

Csikszentmihalyi, M. (1991). *Flow: The Psychology of Optimal Experience.* New York: Harper Collins Publishers.

DuHamel, R. (1962). *Government of Canada Style Manual for Writers and Editors.* Ottawa: Queens Printer and Controller of Stationery.

Fiore, N. A. (1989). *The Now Habit.* New York: St. Martin's Press.

Francik, E. and Akagi, K. (1989). Designing a computer pencil and tablet for handwriting. *Proceedings of Human Factors Society 33rd Annual Meeting,* pp. 445-449.

Goldberg, D. and Goodisman, A. (1991). Stylus user interfaces for manipulating text. *Proceedings of the ACM SIGRAPH and SIGCHI Symposium on User Interface Software and Technology,* November 11-13, Hilton Head, South Carolina, pp. 127-135. ACM.

Goodisman, A. (1991). *A Stylus-Based User Interface for Text: Entry and Editing.* Bachelor of Science and Master of Science thesis, Massachusetts Institute of Technology.

Hardock, G. (1991). Design Issues for Line Driven Text Editing / Annotation Systems. *Proceedings of Graphics Interface '91,* June, Calgary, pp. 77-84.

Kim, J. (1988). On-Line Gesture Recognition by Feature Analysis. *Proceedings of Vision Interface '88,* June 6-10, pp. 51-55.

Kosarek, J. L., Lindstrom, T. L., Ensor, J. R. and Ahuja, S. R. (1990). A Multi-User Document Review Tool. *Proceedings of Multi-User Interfaces and Applications*, pp. 207 - 215. Elsevier Science Publishers.

Kurtenbach, G. and Buxton, W. (1991a). GEdit: a testbed for editing by contiguous gesture. *SIGCHI Bulletin*. pp. 22-26.

Kurtenbach, G. and Buxton, W. (1991b). Issues in combining marking and direct manipulation techniques. *Proceedings of the ACM SIGRAPH and SIGCHI Symposium on User Interface Software and Technology*, November 11-13, Hilton Head, South Carolina, pp. 137-144. ACM.

Kurtenbach, G. and Buxton, W. (1993). The limits of expert performance using hierarchic marking menus. To appear in *Proceedings of INTERCHI '93*, May, Amsterdam.

Levine, S. R. and Ehrlich, S. F. (in press). The Freestyle system: a design perspective. In A. Klinger (Ed.), *Human-Machine Interactive Systems*. New York: Plenum Press.

MacKenzie, I. S. (1992). Movement time prediction in human-computer interfaces. *Proceeding of Graphics Interface '92*, pp. 140-150. Toronto: Canadian Information Processing Society.

Pencept. (1986). *Penpad 310 User's Guide* . Pencept Inc., Waltam, Massachusetts.

Perkins, R., Blatt, L., Workman, D. and Ehrlich, S. (1989). Iterative tutorial design in the product development cycle. *Proceedings of Human Factors Society 33rd Annual Meeting*, pp. 268-272.

Posner, I. (1991). *A Study Of Collaborative Writing*. Master of Science thesis, University of Toronto.

Rhyne, J. R. (1987). Dialogue management for gestural interfaces. *SIGGRAPH Computer Graphics*, *21*(2), pp. 137-142.

Rubine, D. H. (1990). *The automatic recognition of gestures*. Doctoral Disertation, Carnegie Mellon University.

Tappert, C. C., Suen, C. Y. and Wakahara, T. (1990). The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *12*, pp. 787-808.

Ward, J. R. and Blesser, B. (1985). Interactive recognition of handprinted characters for computer input. *IEEE Computer Graphics and Applications*, 5(9), pp. 24-37.

Welbourn, L. K. and Whitrow, R. J. (1988). A gesture based text editor. *People and Computers IV — Proceedings of the Fourth Conference of the British Computer Society Human-Computer Interaction Specialist Group*, Cambridge, pp. 363-371. Cambridge University Press.

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

Wolf, C. G., Rhyne, J. R. and Ellozy, H. A. (1989). The Paper-Like Interface. In G. Salvendy and M. J. Smith (Ed.), *Designing on Using Human Computer Interface and Knowledge-Based Systems* (pp. 494-501). Amsterdam: Elsevier Science.

Appendix A

# MATE Usability Study: Documents

This appendix contains the following documents used in the user testing experiment:

- Introductory page
- Training instructions
- Training document
- Task instructions
- Task document
- Questionnaire
- Interview questions

## A.1 Introductory Page

**MATE** is an application designed to support collaborative writing. To do this, MATE can be used in 3 different modes. MATE's first mode is as a **text editor**. This is to be used by the document's creator / principle author. The primary author then electronically sends this document to the other collaborators, authors, editors, proofreaders, etc. These proofreaders then use MATE in its second mode of operation, that of an **annotation tool**. In this mode the proofreaders mark up the document just as one would mark up a piece of paper. The proofreaders then electronically send the annotated document back to the principle author. The principle author then uses MATE in its third mode, **incorporation tool**, to incorporate the suggested editing changes. In this mode, the document is displayed in 2 windows. The left window is like the annotation mode and shows the annotated document. The right window is similar to the text editor and shows the current version of the document. The principle author can incorporate the annotations either by editing the document directly in the edit (right) window, or by simply selecting the annotations in the annotation (left) window.

For your tasks you will be using MATE in view / incorporate mode to incorporate the annotations in the document given to you.

## A.2 Training Instructions

Start MATE by typing **training**.

This will start the MATE program in incorporate mode and load in a marked-up text document. You will see a document displayed in 2 windows. The window on the left will have various editing marks in it. The text in this window will not change during the training session. The window on the right will initially show the same document as the left window. However, as editing tasks are performed the document in the right window will change to show the current version of the document.

*Feel free to ask any questions during this training session.*

### 1. The Commands

You will see 4 editing marks in the left window. They are **delete, move, broken move** and insert. For each of these do the following:

**1.1** Select the annotation by moving the pen's cursor on top of the mark, and then tap the pen on the graphics tablet. Note that you must be pointing to the mark portion of the insert annotation to select it. The annotation will become thinner to indicate that it has been incorporated. Observe the effects of the incorporation in the edit window. Note that the document in the annotation window does not change.

**1.2** Re-select the annotation. This will undo the incorporation of the annotation. Notice that the mark becomes thick again. Observe the effects in the edit window.

**1.3** Make the corresponding editing mark in the edit window. The command will be executed once the pen is lifted from the graphics tablet. The following is a description for specifying the various commands:

**Delete:** a horizontal mark over the text to be deleted.

**Move:** a circle around the text to be moved, followed by a line to the destination. Note that the circle and line should be one continuous mark.

**Insert:** the insert mark is a "^" or "v". Both marks are made from left to right. If the system recognizes the insert mark, it will pop-up a text window. If the text window does not pop-up, MATE has not recognized your insert mark and you must try again. Type the text you wish to insert, and then press the Insert key. If you want to cancel the insert, press the Escape key.

**Broken Move:** This is used for move commands for which the source and destination are far apart. There are two parts. The first starts the same as the move command i.e., a circle around the source text followed by a line. But in this case the line is drawn to the margin. MATE will recognize this as a broken move command and will allow you to choose which symbol you want using marking menus (described below).

The second half of the broken move command can be made after navigating around in the document. Simply pen down (usually in the margin) and draw a line to the destination. The symbol will be placed automatically when you start the mark. Note that you should only navigate between the two halves of the broken move command.

**Aside: Using Marking Menus:**

Marking menus are special pop-up menus. First, the menu items are arranged in a circle. To select an item you move the cursor into the appropriate sector of the circle and lift the pen. Note that the sectors of the circle extend to the edge of the screen. To not make a selection move the cursor to the centre area of the menu. This is a dead-zone for which no menu item will be selected.

It is not necessary to wait for the menu to pop-up. You might have noticed a line connecting the centre of the menu to the cursor. You can make a selection by drawing a similar mark before the menu appears. The benefits of marking ahead will become more apparent when using marking menus for the navigation commands.

**1.4** Undo your edit by selecting the undo button on the screen with the mouse. This will undo your last editing command.

*Repeat any of the above steps until you are comfortable with the command set.*

## 2. Moving through the document (Navigation)

### Basic Navigation:

Basic navigation is accomplished in navigation mode. To be in navigation mode press and hold the button on the pen. The basic navigation commands are **next page, previous page**, and **shift up/down several lines**. A good way to think of these navigation commands is to consider moving or pushing a piece of paper. Page movements are fast, like flicking a page, whereas, shifting lines is slower, like pushing the paper. To move the document to the next page, you "flick" the document forward / upward. To move the document to the previous page, you flick the document backward / downward. For shifting several lines, consider the movement to be pushing the document. The length of the line indicates how far you are pushing the document. The shifting lines navigation commands are part of a marking menu, and can be used exactly as outlined in the description of marking menus.

**2.1** Use the page flicks to move each window to the bottom of the document and back to the top again.

**2.2** Use the "move line to ..." commands to move each window to the bottom of the document and back to the top again.

### Linked Navigation

The page flicks and line shifting can be done separately in each window or they can be linked so that both versions of the document move at once. To do this, start a page flick or line shift by drawing the appropriate up or down navigation mark and then without lifting the pen make an "L" shape by drawing either to the left or to the right. Note that linked movement only works while marking. If the menu appears, only unlinked movement will occur.

**2.3** Use the linked page flicks to move each window to the bottom of the document and back to the top again.

**2.4** Use the linked "move line to ..." commands to move each window to the bottom of the document and back to the top again.

*Repeat any of the above steps until you are comfortable with the command set.*

## 3. Advanced Navigation Techniques

Initially the 2 documents are the same and there should be no difficulties. But as changes are made to the edited document, the 2 versions may become significantly different. The advanced navigation commands are designed to aid in understanding the relationship between the 2 versions of the document. These commands are **GoTo Word, GoTo Annotation**, and **GoBack**. GoTo word and GoBack are part of the navigation command marking menu described above. GoTo Annotation and GoBack are part of another marking menu that acts on annotations. To use the marking menu for annotations, point to the annotation you wish to do the command to and hold the pen down. The marking menu for annotations will appear, containing the options "Hide", "GoTo", "Erase", and "GoBack".

GoTo word does one of 2 things. If the word on which you penned down over, appears in both windows, the word will be highlighted in the other window. For example, using the basic navigation commands move the edit window so it is not at the top of the document but "MATE2" still appears. Specify GoTo word on the word "MATE2" in the annotation window. The word "MATE2" in the edit window will be highlighted. If another GoTo word command is specified, the 2 windows will be lined up, i.e. the lines on which that word appears are side by side. To see this specify GoTo word on the word "MATE2" in the annotation window again.

If the word on which you specified the GoTo word is not in the other window, the 2 windows will be lined up with 1 GoTo word. For example, using the basic navigation commands move the edit window so that "MATE2" does not appear. Now specify GoTo word on the word "MATE2" in the annotation window again. This time the 2 windows are lined up with one GoTo word.

GoBack simply undoes the GoTo Word and GoTo Annotation commands.

**3.1** Navigate to the top of the annotation window using the basic navigation methods. Then use the basic navigation to navigate through the annotation window. For every occurrence of the word "line-up" use the GoTo word command to line-up the 2 windows.

GoTo Annotation is similar to GoTo Word, except that it highlights the source text and / or destination of the command specified by the annotation. If the source text and destination cannot be shown at the same time, as is usually the case with the broken move command, GoTo Annotation will alternate between lining up the source text and the destination.

**3.2** Navigate to the top of the annotation window using the basic navigation methods. Then use the basic navigation to navigate through the annotation window. Find each occurrence of an annotation with the word "line-up" in it. Then use the GoTo annotation command to observe which text would be affected by the annotation.

*Repeat any of the above steps until you are comfortable with the command set.*

## 4. Overall Training Task

Navigate to the top of the document in both windows.

Incorporate all of the annotations using a combination of selecting the annotations in the annotation window and directly editing the document in the edit window. During this task, try using all of the navigation methods so that you are comfortable using them.

Repeat any of the commands as often as you like until you are comfortable with them.

## 5 Ending the Training Session

Once you are finished with the training task, do the following.

**5.1** Save the training document by pressing the Document Save button with the mouse.

**5.2** Press the Quit button with the mouse to exit the program.

***This concludes the training portion of this experiment.***

## A.3 Training Document

Welcome to MATE. During this training session, you will be modifying this document.

This is an example of a delete command.

This is an example of a move command to here from there.

This is an example of a long distance move command using markers from there.

This is an command example.

To here before MATE2.

This is the first occurence of line-up (1).

You have been viewing while in the view / incorporate mode this document.

When you GoTo a delete annotation the windows and you see the text to be deleted.

If MATE does not your insert mark, you will not be with the pop-up text entry box.

Don't you think that this paragraph should go somewhere else.

When you GoTo a broken move annotation, the windows line-up (3) so that you see the source text. A second GoTo shows the destination.

MATE supports collaborative writing asynchronous.

This is the end of the user training sample document. Have a nice day.

When you GoTo a move annotation, the windows line-up (4) and you see the source text and the destination of the move command.

Sometimes a delete command may cover many words. At other times it may only apply to a single word.

This is where the destination of one of the broken move annotations resides.

When you Goto an insert annotation, the windows and you see the point where the text will be inserted.

## A.4 Task Instructions

**Scenario:** Imagine that you are a newspaper reporter. You have written a story entitled "Holidays from Hell", which you sent to 2 editors for proofreading. They have sent their comments and corrections to you. Your task is to edit the document to your satisfaction, keeping the editors comments in mind and using them where appropriate.

**1.** Start MATE by typing **task** and pressing <return>.

This will start the MATE program in incorporate mode and load in a marked-up text document. You will see a document displayed in 2 windows.

**2.** Edit the document to your satisfaction. You may use any of the annotations given or make your own editorial changes.

**3** Once you are finished with the task, do the following.

**3.1** Save the document by pressing the Document Save button with the mouse.

**3.2** Press the Quit button with the mouse to exit the program.

***This concludes the main task portion of this experiment.***

## A.5 Task Document

HOLIDAYS FROM HELL

Everything went wrong on Isherwood's trip to California, and it wasn't at all amusing at the time, he can laugh about the experience now. On the other hand, Harry Hoyle's first vacation turned into a real-life tragedy. Fortunately, the nightmare permanently dampen his enthuasm for travelling.

Do you Remember the Murphy Awards?

Back in January, we readers to write and tell us about travel experiences that got so badly fouled up that Murphy's Law ("anything that can go wrong will") seems to be at work. Steve Isherwood of Islington rose to the occasion with "California Nightmare."

"The plane began its descent into the Bay Area, and suddenly my trip became a nightmare. I've always enjoyed going to California on business and as I dozed on the 767 to San Francisco, my thoughts were of surfing, beaches and bikinis."

"And then the vomit came. Lot's of it. Those little bags the airline provides didn't catch much. But my wrinkle-resistant dress suit caught a lot. The plane veered straight up. The heads of the passengers shot backwards into our seats. The captain's voice apologized over the loudspeaker, `Sorry for any inconvenience. We had to avoid another plane on the runway.'"

And the nightmare trip was just beginning.

Isherwood writes: "I woke up at four in the morning. The room was rocking. I was sure the bed been wedged against the hall door when I went to sleep. Just then, the ground rumbled.

"Sure enough, it was an earthquake. Not that tragic big one when the Bay Bridge and a highway collapsed, but big enough to ensure a sleepless night of terror."

In the morning, he joined a business contact again for breakfast.

"Just as our eggs eggs arrived, the rumbled and the tables in the room collapsed like dominoes. It was an aftershock."

"So much for my only remaining business suit."

"The cabbie was none too keen to take me to my hotel in San Francisco."

"In Los Angeles, I found that my prepaid hotel had gone out of business. I passed my second sleepless night in a run-down hotel, paid money to park my rental car, and drove through a five-car pileup only to find out that my tire had collapsed."

"To soothe my nerves, I decided to check out the beach. But when I got there, I couldn't see one bikini or surfboard."

"The beach was in fog."

* * * * * * *

Isherwood and Hoyle are joint winners of Murphy Awards, and each get $200 for their tales of extraordinarily bad trips.

Harry Hoyle, a Peel Region police officer who lives in Georgetown, wrote to us about what happened to him on May 9, 1977, "a day forever burned in our memories."

The Hoyles arrived in Amsterdam, on their first European vacation. They judged their hotel, just a block from the royal palace on Dam Square, to be "beautiful." Since they planned on staying a few days, Harry sent three jeans and two shirts to the hotel laundry.

But this perfectly routine chore would mark the beginning of a tragic chain of events.

The shirts came quietly back from the laundry but the jeans didn't.

"We'd planned to check out early on Saturday," says Harry, "but since the stupid management refused us to compensate for the loss of the jeans, we decicided we had no choice but to stay put until the laundry re-opened on Monday.

"It was either that or run around Europe in my knickers."

They never got the jeans.

Let Harry tell the story:

"On Monday morning, we awakened by smoke and roaring flames. There was a terrible fire at the hotel.

Harry and Laura Hoyle survived, but 43 other hotel guests perished.

"We were able to help six Swedish seniors to safety before jumping out a third-floor window ourselves."

After two months in a burn centre in The Netherlands, they came back to Canada for two more months in hospital. Both sustained severe multiple injuries, including broken bones and third-degree burns requiring skin grafts.

After their recovery, the Hoyles indulge their love of travel, but, says Harry, "now I always insist on a room no higher than the second floor."

Sadly, Laura Hoyle died last year.

Says Harry: "When I travel I carry a rope in my backpack and follow stairwells all the way down to see where the fire exits go."

"And, would you believe, many fire 'exits' in European hotels aren't exits at all. They're just for show, they don't lead anywhere!"

## A.6 Questionnaire

### Editing Text and Incorporating Annotations

In the questions which follow, please circle the category which best represents how you feel about your experience in moving through the document with (1)=strongly disagree, (2)=disagree, (3)=don't agree or disagree, (4)=agree, (5)=strongly agree, and (6)=not applicable or N/A.

1. I selected annotations for incorporation rather than do the edits directly in the edit window because I had difficulty in making editing marks which the system could understand properly.

| | | | | | |
|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 |
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

2. I preferred to do the edits directly in the edit window even when I could have chosen an annotation for incorporation to do the same edit.

| | | | | | |
|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 |
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

3. I preferred using the *undo last* command over the *undo by selection* command.

| | | | | | |
|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 |
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

4. I could do the tasks with the given commands (move, delete, insert).

| | | | | | |
|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 |
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

In the questions which follow, please choose the category which best represents your use of MATE.

5. Please indicate how easy / difficult it was to perform the tasks by editing directly (i.e. specifying editing commands in the edit window), and by selecting annotations for incorporation.

| | very difficult | moderately difficult | somewhat difficult | somewhat easy | moderately easy | very easy |
|---|---|---|---|---|---|---|
| edit directly | | | | | | |
| selecting annotations | | | | | | |

6. Please indicate how easy / difficult it was for you to specify the following commands in a way which the system recognized properly.

| | very difficult | moderately difficult | somewhat difficult | somewhat easy | moderately easy | very easy |
|---|---|---|---|---|---|---|
| delete | | | | | | |
| move | | | | | | |
| broken move | | | | | | |
| insert | | | | | | |

7. Overall, how would you rate the system as a tool for incorporating annotations and for editing text? (please tick appropriate category)

| Very Satisfactory | Moderately Satisfactory | Neutral | Moderately Unsatisfactory | Very Unsatisfactory |
|---|---|---|---|---|
| | | | | |

8. What did you like most regarding any of the incorporation / editing features?

9. What did you like least regarding any of the incorporation / editing features?

## Navigation

In the questions which follow, please circle the category which best represents how you feel about your experience in moving through the document with (1)=strongly disagree, (2)=disagree, (3)=don't agree or disagree, (4)=agree, (5)=strongly agree, and (6)=not applicable or N/A.

1. *Page flicks* did what I initially expected them to do.

| | | | | | |
|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 |
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

2. In general, I would prefer to use a scrollbar instead of *page flicks* to move up / down a page in a document.

| | | | | | |
|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 |
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

3. The analogy of turning pages helped my understanding of *page flicks.*

| | | | | | |
|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 |
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

4. *Move line to here* did what I initially expected it to do.

| | | | | | |
|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 |
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

5. In general, I would prefer to use a scrollbar instead of *move line to here* to make small movements in a document.

| | | | | | |
|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 |
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

6. The analogy of pushing a piece of paper helped my understanding of *move line to here.*

| | | | | | |
|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 |
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

7. *Linked navigation commands* did what I initially expected them to do.

| | | | | | |
|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 |
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

8. *Goto (Navigation menu)* did what I initially expected it to do.

| | | | | | |
|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 |
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

9. *Goto (Annotation menu)* did what I initially expected it to do.

| | | | | | |
|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 |
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

10. *GoBack* did what I initially expected it to do.

| 1. | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

11. The *GoBack* command was useful to temporarily examine a section of the document and then jump back to where I was working.

| 1. | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| strongly disagree | disagree | neither agree or disagree | agree | strongly agree | N/A |

In the questions which follow, please circle the category which best represents your use of MATE in moving through the document.

12. Please indicate how easy / difficult it was for you to specify the following navigation commands in a way which the system recognized properly.

| | very difficult | moderately difficult | somewhat difficult | somewhat easy | moderately easy | very easy |
|---|---|---|---|---|---|---|
| page flick | | | | | | |
| move line to here | | | | | | |
| linked navigation | | | | | | |

13. Please indicate how useful the *Goto (Navigation menu)* command was for the following purposes.

| | not at all useful | not too useful | Don't know not sure | fairly useful | very useful |
|---|---|---|---|---|---|
| lining up the windows | | | | | |
| navigating through the document | | | | | |
| as an aid in understanding the relationship between the two versions of the document | | | | | |

14. Please indicate how useful the *Goto (Annotation menu)* command was for the following purposes.

| | not at all useful | not too useful | Don't know not sure | fairly useful | very useful |
|---|---|---|---|---|---|
| lining up the windows | | | | | |
| navigating through the document | | | | | |
| as an aid in understanding the relationship between the two versions of the document | | | | | |
| as an aid in understanding what text would be affected by the annotations | | | | | |

15. Overall, how would you rate the system in terms of navigation? (please tick appropriate category)

| Very Satisfactory | Moderately Satisfactory | Neutral | Moderately Unsatisfactory | Very Unsatisfactory |
|---|---|---|---|---|
| | | | | |

16. What did you like most regarding any of the navigation features? Why?

17. What did you like least regarding any of the navigation features? Why?

## SYSTEM USABILITY PROBLEMS

Please check the appropriate box and feel free to make any comments.

| | no problems | minor problems | major problems | comments |
|---|---|---|---|---|
| 1. Working out how to use the system | | | | |
| 2. Lack of guidance on how to use the system | | | | |
| 3. Understanding how to carry out the tasks | | | | |
| 4. Knowing what to do next | | | | |
| 5. Understanding how the two windows are related | | | | |
| 6. Losing track of where you are in the system or of what you are doing or have done | | | | |
| 7. Using the pen in navigation mode (i.e. using the pen while holding the button down) | | | | |
| 8. Using the pen (in general) | | | | |

| | no problems | minor problems | major problems | comments |
|---|---|---|---|---|
| 9. Unexpected actions by the system | | | | |
| 10. Having to spend too much time navigating through the document | | | | |
| 11. Having the proper tools to carry out the functions necessary to edit a document | | | | |
| 12. Making marks which were recognizable by the system | | | | |

## USER SATISFACTION

On a scale of 1–5 please circle the number which best represents your reaction to each aspect of MATE. Please feel free to write any comments you may have anywhere on the questionnaire.

1. How easy to use is the system?

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | X |
| difficult | | | | easy | no opinion |

2. How satisfying is the system to use?

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | X |
| frustrating | | | | satisfying | no opinion |

3. How interesting is the system?

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | X |
| dull | | | | stimulating | no opinion |

4. How flexible / rigid is the system?

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | X |
| rigid | | | | flexible | no opinion |

5. What is your overall opinion of the system?

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | X |
| terrible | | | | wonderful | no opinion |

6. Is the system easy/difficult to learn?

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | X |
| difficult | | | | easy | no opinion |

7. Could you explore features by trial and error?

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | X |
| difficult | | | | easy | no opinion |

8. Can tasks be performed in a straightforward manner?

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | X |
| never | | | | always | no opinion |

9. Are the names and use of commands easy/difficult to remember?

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | X |
| difficult | | | | easy | no opinion |

10. Is it easy/difficult to understand how to carry out the tasks?

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | X |
| difficult | | | | easy | no opinion |

Please give any additional comments you would like to make about MATE.

***THANK YOU FOR YOUR HELP!***

## A.7 Interview Questions

1. Where there any questions in the questionnaire which you didn't understand?

2. Was there anything you wanted to say that was missing from the questionnaire?

3. Was there anything about the navigation which you really liked or disliked?

4. Was there anything about the system that confused you?

5. Was there anything about the system that was difficult to do?

6. Was there anything about the system that you really liked?

7. Was there anything about the system that you really did not like?

8. What situations do you think a system such as MATE would benefit, as an incorporation tool for marked-up documents, as an annotation tool, or as a text editor?

9. Is there anything else which you would like to comment on?

# Appendix B

# MATE Usability Study: Results

## B.1 Training Observations

The following are observations from the videos of the screen and subject as well as the log files recorded during the training session. Only minor modifications were made to the training procedure after pilot 2, and thus pilot 2's data is also noted here.

*Pilot 2*

Training time: 45 minutes

- had problems in drawing a recognizable insert caret.
- was initially confused about direction of page flicks.
- suggested that some visual indication would be useful to tell when MATE is in navigation mode.
- Move line to here was not clear initially.
- had problem with linked page flick – not able to move fast enough to register as a page flick.
- System response was very slow – half a minute to respond to a linked movement.
- would have liked to get to the top quickly. Page flicks and pushes do not provide an easy way to do this.
- had difficulty in using button on pen; resorted to using the mouse.
- Bug in marking menus. Press and hold in place does not bring menu up. The pen must be moved slightly.
- noted that the editing commands are fairly easy to do, but the navigating is hard. Also subject sometimes forgot to push the button on the pen to enter navigation mode.

- would like to be able to make annotations in the Edit View, or have an editing mark stored as an annotation in the Annotation View.

*Subject 1*
Training time: 45 minutes

- had no problems until navigation. A page push command took a minute to execute.
- Button on pen broke. User had to use the mouse (using the middle button on the mouse to enter navigation mode) to navigate.
- was confused between Goto in annotation menu and Goto in navigation menu. Subject suggested that they be distinctly labeled.
- experienced some confusion with the direction of page flicks.

*Subject 2*
Training time: 30 minutes

- had a lot of difficulty in making a recognizable insert caret. Several times it was mistaken as a delete mark.
- System hung for over a minute while attempting to do a goto command.
- preferred using the mouse for navigation commands, especially for linked navigation.

*Subject 3*
Training time: 38 minutes

- experienced some confusion after accidentally erasing a mark.
- noted confusion that a different action – annotation menu – occurs when pointing to a mark versus not pointing to a mark – drawing.
- System hung. Had to restart the program.
- commented that the button is difficult to use. It would be much easier if you had a little area in which to specify the navigation commands.
- commented that one has to remember to push the button on pen before you put the pen down.
- Bug in program. Subject made a move to placeholder mark in the Edit window, and then started to draw in the Annotation window. The system mistook the mark in the Annotation window as a move from placeholder mark.

- was annoyed that he cannot use the pen in the scroll bar area to navigate (using the standard X window navigation techniques.).

## B.2 Task Observations

The following are observations from the videos of the screen and subject as well as the log files recorded during the task session. Only minor modifications were made to the task after pilot 2, and thus pilot 2's data is also noted here.

*Pilot 2*

- used mouse in scrollbar and in navigation mode to navigate.
- The file was slightly corrupted due to accidentally using a previously edited file. The effect was that some of the annotations were slightly shifted from the text, thus not allowing the annotation to be directly incorporated. In most cases the meaning of the annotation was still clear to the user.
- made five attempts to make a recognizable insert mark before making one which the system recognized. On the average, the recognition rate was far below 50 %.
- used goto mark to show the parameters of a delete mark.
- experienced some confusion about the goto command.
- attempted to make a short delete mark for deleting a single character three times before making one the system recognized.
- System hung for a minute on a page move command.
- wanted to make a side note to himself in the Edit window. Initially tried to make a move annotation, then resorted to inserting some descriptive text.

*Subject 1*

- used mouse for navigation and pen for everything else. This was because the button on the pen had broken during the training session. The subject held the pen in her hand when using the mouse.
- A bug in the program occurred when the subject tried a move command. Several lines of text were lost and two other lines were duplicated. This obviously confused the subject. She did not think about using the undo last command, which she never used. But it is unclear whether the undo last would have corrected the problem. The remaining duplicated text caused some confusion when operations were performed on

it.

- Other than the above difficulties the user appeared to have little difficulty
- The file was corrupted slightly, i.e. the text file and markup files did not match after the second page as a previously edited version of the text file was accidentally used.
- tried inserting several lines of text, but the insert box could only show one line of text. To see what she was entering she entered return characters. This caused more problems as the main text windows have word wrapping and the insert box does not.
- The system hung for half a minute for a linked page flick.
- An insert was mis-recognized as a delete mark. Overall the recognition of the insert mark was not too bad, but there were a few cases in which the user had to make several attempts at drawing a recognizable insert mark.
- often wanted to append text to words rather than enter entire words. As MATE was in word insert mode for the user testing, the subjects never has to worry about entering spaces around the inserted text, MATE would add the spaces where necessary. But this caused problems when the user was not inserting words but suffixes. The subject had to delete the extraneous spaces.
- had little difficulty in specifying all of the navigation commands with the mouse.

*Subject 2*

- had no major problems, being able to do all of the commands.
- used GoBack once, which caused no effect – neither view changed.
- used Undo last several times.
- deleted several lines of a paragraph using several delete commands. It would be easier for the user if MATE had a way of deleting several lines at once.
- tried pressing the link and unlink buttons located below the scroll bars. These buttons were not active and caused confusion to the user. The user did user both linked and unlinked navigation commands but mainly unlinked.
- used the Goto Annotation command twice to see the various parameters of a broken move command. When MATE jumped to the destination and highlighted it, the subject noticed that part of the source text was visible. But when the user pushed the page to see more source text, the highlighting disappeared, and the user lost track of where she was in the document. This brings about the issue of how MATE should adjust the

document for the Goto Annotation command.

- had some difficulty in drawing a recognizable mark to delete a single character. This is due to a minimum length criteria for recognizing marks.

*Subject 3*

- selected a mark for incorporation but it disappeared, and was not incorporated. Bug in program, unsure of cause.
- forgot how to do the insert command, had to refer back to training notes.
- A move command was misinterpreted as a move to placeholder as the user went too far into the margin.
- made an incorrect statement that insert does not seem to work at non-spaces. The problem is the poor recognition of the insert mark.
- The small non-scrollable text entry box for insert caused a problem when the subject tried to enter a paragraph of text. Similar to subject 1, this subject used return characters to see the text he was entering, which causes problems when the text is inserted into the document.
- asked if there is a way to change or replace a word. The answer is not in a single step, the user must delete and insert the text. Subject mentions that it would be convenient if words could be changed.
- asked if there is any way to cross out more than one line at a time.
- System hung on a page flick or page push command.
- commented that: it seems like in part the effectiveness depends upon whether or not you are going to be accepting the annotations. If you accept a lot of changes then it's very easy, but if you want to think about the changes then write it over again yourself then it becomes harder.
- used the mouse in the scroll-bar for some navigation.
- made several unrecognizable marks in a row. The "unrecognizable command" popup box, causes a delay in accepting the pen input which causes the next mark to be unrecognizable, thus initiating a chain reaction of unrecognizable marks.
- had difficulty in deleting a single character.
- wanted to insert the word "a" between two words which didn't have a space between them. The system did not generate the spaces that the user wanted as it assumed that he

was modifying a word. Subject commented that the system never does what you want it to.

- System hung, had to restart the program.
- did use undo last once.

## B.3 Questionnaire Results

This section gives the results from the questionnaire. The following three tables summarize questions which were rated on a scale. Table B.1 summarizes questions concerning ease and difficulty, table B.2 summarizes questions on a 5 point scale, and table B.3 summarizes questions concerning usefulness.

| Section | # | Question | 1 | 2 | 3 | 4 | 5 | 6 | Avg |
|---|---|---|---|---|---|---|---|---|---|
| Editing Text and Incorporating Annotations | 5 | Ease in editing directly and in selecting annotations | | | | | | | |
| | | edit directly | 1 | 0 | 0 | 1 | 1 | 0 | 3.33 |
| | | selecting annotations | 0 | 1 | 0 | 0 | 0 | 1 | 4.00 |
| | 6 | Ease in specifying recognizable commands | | | | | | | |
| | | delete | 0 | 1 | 0 | 0 | 2 | 0 | 4.00 |
| | | move | 0 | 0 | 0 | 1 | 2 | 0 | 4.67 |
| | | broken move | 0 | 0 | 0 | 0 | 1 | 0 | 5.00 |
| | | insert | 0 | 0 | 3 | 0 | 0 | 0 | 3.00 |
| Navigation | 12 | Ease in specifying recognizable navigation commands | | | | | | | |
| | | page flick | 1 | 0 | 1 | 0 | 0 | 1 | 3.33 |
| | | move line to here | 0 | 0 | 0 | 0 | 2 | 1 | 5.33 |
| | | linked navigation | 0 | 0 | 0 | 2 | 1 | 0 | 4.33 |

**Table B.1**

*Summary of questions on ease / difficulty of use. The scale ranges from 1 – very difficult to 6 – very easy.*

| Section | # | Question | 1 | 2 | 3 | 4 | 5 | n/a | Avg |
|---|---|---|---|---|---|---|---|---|---|
| Editing Text and Incorporating Annotations | 1 | prefer selecting annotations | 0 | 2 | 0 | 1 | 1 | 0 | 3.25 |
| | 2 | prefer editing directly | 0 | 2 | 1 | 1 | 0 | 0 | 2.75 |
| | 3 | prefer undo last | 0 | 0 | 2 | 0 | 0 | 2 | 3.00 |
| | 4 | sufficient command set | 1 | 0 | 0 | 3 | 0 | 0 | 3.25 |
| | 7 | Satisfaction | 0 | 1 | 0 | 2 | 1 | 0 | 3.75 |
| Navigation | 1 | Page flicks acted as expected | 0 | 1 | 1 | 1 | 0 | 1 | 3.00 |
| | 2 | prefer scrollbar over page flick | 0 | 0 | 1 | 3 | 0 | 0 | 3.75 |
| | 3 | page turning analogy helped | 1 | 0 | 2 | 0 | 1 | 0 | 3.00 |
| | 4 | Move line to here acted as expected | 0 | 1 | 1 | 2 | 0 | 0 | 3.25 |
| | 5 | prefer scrollbar over move line to here | 0 | 1 | 2 | 1 | 0 | 0 | 3.00 |
| | 6 | page push analogy helped | 0 | 0 | 0 | 1 | 3 | 0 | 4.75 |
| | 7 | linked navigation acted as expected | 0 | 1 | 0 | 2 | 1 | 0 | 3.75 |
| | 8 | Goto (Navigation Menu) acted as expected | 0 | 0 | 0 | 4 | 0 | 0 | 4.00 |
| | 9 | Goto (Annotation Menu) acted as expected | 0 | 0 | 0 | 4 | 0 | 0 | 4.00 |
| | 10 | Goback acted as expected | 0 | 0 | 0 | 2 | 1 | 1 | 4.33 |
| | 11 | Goback useful to temporarily examine section of text | 0 | 0 | 1 | 1 | 1 | 1 | 4.00 |
| | 15 | Satisfaction | 1 | 0 | 0 | 3 | 0 | 0 | 3.25 |
| User Satisfaction | 1 | easy to use | 1 | 0 | 0 | 3 | 0 | 0 | 3.25 |
| | 2 | satisfying | 1 | 0 | 0 | 2 | 0 | 1 | 3.00 |
| | 3 | interesting | 0 | 0 | 1 | 1 | 2 | 0 | 4.25 |
| | 4 | flexible | 1 | 0 | 0 | 0 | 2 | 1 | 3.67 |
| | 5 | overall opinion | 0 | 0 | 1 | 2 | 0 | 1 | 3.67 |
| | 6 | easy to learn | 0 | 0 | 0 | 2 | 2 | 0 | 4.5 |
| | 7 | easy to explore | 0 | 2 | 1 | 1 | 0 | 0 | 2.75 |
| | 8 | straightforward tasks | 0 | 1 | 1 | 2 | 0 | 0 | 3.25 |
| | 9 | easy to remember commands | 0 | 0 | 1 | 1 | 2 | 0 | 4.25 |
| | 10 | easy instructions | 0 | 0 | 0 | 1 | 1 | 2 | 4.5 |

**Table B.2**

*Summary of questions rated on a five point scale.*

| Section | # | Question | 1 | 2 | 3 | 4 | 5 | Avg |
|---|---|---|---|---|---|---|---|---|
| Navigation | 13 | Usefulness of Goto (Navigation Menu) | | | | | | |
| | | lining up the window | 0 | 0 | 3 | 0 | 1 | 3.50 |
| | | navigating through the document | 0 | 0 | 2 | 2 | 0 | 3.50 |
| | | aid in understanding relationship between views | 0 | 1 | 0 | 3 | 0 | 3.50 |
| | 14 | Usefulness of Goto (Annotation Menu) | | | | | | |
| | | lining up the window | 0 | 0 | 4 | 0 | 0 | 3.00 |
| | | navigating through the document | 0 | 1 | 3 | 0 | 0 | 2.75 |
| | | aid in understanding relationship between views | 0 | 0 | 2 | 1 | 1 | 3.75 |
| | | aid in understanding what text would be affected by the annotation | 0 | 0 | 1 | 2 | 1 | 4.00 |

**Table B.3**

*Summary of questions on usefulness. The scale ranges from 1 – not at all useful to 5 – very useful; 3 is don't know or not sure.*

Questions which asked the user for comments are given below.

*Editing and Incorporating Annotations*

8. What did you like most regarding any of the incorporation / editing features?

I found it easy and desirable to issue the incorporate annotation command instead of manually doing the edit.

I liked the basic idea a lot. Like the ability to incorporate editor's annotations without having to re-specify them - except in this example, The inserts were placed to the wrong point and something went wrong. When I tried to "move" a misplaced insert.

The ability to move the windows either together or independently was a help. The pen is easy to use: very intuitive.

Being able to pick text out for deletes easily and could move very easily.

Incorporation of exactly the marked changes was very easy.

It was fun circling things.

9. What did you like least regarding any of the incorporation / editing features?

The insert command did not get recognized well enough.

1) Insert! at points, especially for insert single characters, it would have been much easier to have been able to click in the edit window to show where to insert, and type the text directly in (Mac-like, I guess). Also, for multi-line inserts, would be nice to see them all while typing in.

2) Somehow, I lost several lines of text and eventually had to type them all in. I had been trying to move when I lost them.

3) Inserting words is easy, (except for general insert problems noted above) but more difficult to just add a letter or suffix attached to previous word. Would be nice to have "modify" or "change" ability in edit window, instead of delete then insert. Maybe in rev II?

4) Ability to see more lines of text at a given time would be VERY helpful. These were too few lines to really get the flow of the prose as it were.

a little hard getting insertions but I did get better the more I practiced.

hard to controlling exactly.

** Hard to manipulate 3 input devices at once.

command set was too small and hard to use.

*Navigation*

16. What did you like most regarding any of the navigation features? Why?

The idea of linked navigation is very good.

goto annotation was great, and partially made up for the difficulty with too small pages.

page flicks is cute but probably would prefer scroll bar.

The "move to here" feature was ok, although I did get it backwards once.

17. What did you like least regarding any of the navigation features? Why?

I found it difficult to use the pen when making "flicks" - but using the mouse, I got much better results.

cursor should change its shape when the pen is pressed (navigation commands).

page flick & move line here well explained and made sense in that context, but still had trouble using them correctly, because I expected to virtually grab & pull the text to move it, instead of virtually grabbing & pushing the paper. (Does this make sense? I expected them to move the opposite direction sometimes, until I thought about it.)

never was able to get the windows aligned, link seemed to be working funny.

Couldn't move to a position and insert characters there.

Hard to keep your eyes on two documents at once. Having one working document is an advance, splitting it into two is a step backwards in technology. You could have one evolving document with a "view original" button.

*System Usability Problems*

| | no problems | minor problems | major problems | comments |
|---|---|---|---|---|
| 1. Working out how to use the system | 1 | 3 | - | Mostly using the pen. Had a tough time to push the pen button for navigating.<br>Had to remember difference between goto annotation and goto word - should be some better mnemonic & distinction between menus. |
| 2. Lack of guidance on how to use the system | 4 | - | - | Gary answered my questions<br>Good instructions. Spell out numbers <= twelve! |
| 3. Understanding how to carry out the tasks | 3 | 1 | - | Forgot about "undo"! But good that there are more than one way to get a given effect. |
| 4. Knowing what to do next | 3 | 1 | - | |
| 5. Understanding how the two windows are related | 4 | - | - | goto annotation helped. Bigger windows (longer swatch of text visible) would help a LOT! |
| 6. Losing track of where you are in the system or of what you are doing or have done | - | 4 | - | see "Bigger windows" above.<br>For the broken edits trying to figure out where the text was going<br>took so long to make trivial changes that I forgot what the main editorial task was. |
| 7. Using the pen in navigation mode (i.e. using the pen while holding the button down) | - | 2 | 1 | Remembering to push the pen button.<br>N/A (broke) (or is breaking "major prob" ??!)<br>button didn't always register<br>physically difficult |

| | | | | |
|---|---|---|---|---|
| 8. Using the pen (in general) | - | 3 | - | having to make an insertion between 2 characters - difficult to get needed precision.<br>ditto!<br>hard to get hand eye coordination since your eyes are on the screen |
| 9. Unexpected actions by the system | - | 2 | 2 | unrecognized commands ^, -<br>lost things (text) when trying to insert doubled lines, then lost those. WAY confusing! (see video at ~ "eggs eggs" para)<br>slow |
| 10. Having to spend too much time navigating through the document | 1 | 3 | - | needed more text showing<br>was ok once I got used to using scroll-bars with the left hand |
| 11. Having the proper tools to carry out the functions necessary to edit a document | 3 | - | 1 | Wanted to be able to make notes in margin or on text which were not mistaken for editing commands<br>It would have been easier with text editor commands, for the right-hand document |
| 12. Making marks which were recognizable by the system | 1 | 3 | - | unrecognized commands ^, -<br>insert worked relatively well for me, but still some difficulties.<br>just insert and that got better with time<br>took a little getting used to |

*User Satisfaction*

Please give any additional comments you would like to make about MATE.

Neat idea.

pen interface very natural, except the button on the pen (which broke anyway).

Good to have alternate (mouse) interface!

May I have a copy?

An interesting idea and a substantial undertaking for a Master's project.

I think it would be easier to use if there were just one document displayed.

There may be a limit to how many commands you can squeeze into a pen motion!

## B.4 Interview Results

The following is a condensed transcription of the interviews. P1 and P2 denote responses from the two pilot subjects and S1, S2, and S3 denote responses from the three subjects. In some cases the interviewer's comments are shown to give context to the subject's responses; interviewer comments are denoted by I. At times either the subject or interviewer performed an action in MATE to add to their comments; these actions are indicated in parentheses.

1. Were there any questions in the questionnaire which you didn't understand?

(P2) Yes. (Major changed were made to the questionnaire after the second pilot subject.)

(S1) I wasn't able to answer question 5. It was a depends question. if everything had been perfect. (The text file was corrupted, and the annotations did not line up with the text, therefore the user couldn't directly select any of the annotations.)

2. Was there anything you wanted to say that was missing from the questionnaire?

(S2) Why use a pen that leaves ink on paper?

(S3) I think the questionnaire covers it. I mean I think there are limitations on the media on the pen based idea. I think there are limitations because of the width of the lines, how easy it is to draw a line around a character, or cross out. Those would probably be improved by future technology for pens and screens. For example, if the screen and the tablet were the same. I think the ideas are bigger than the technology that you have. It's an interesting direction. I think a lot depends on what the person's like. A lot of people prefer writing over typing. In this particular case you can only see one line when you are typing (in the insert box).

(S3) I think it would be easier to have only one document. I'm thinking of having one document as the marked up document, and it adjusts as the document changes.

3. Was there anything about the navigation which you really liked or disliked?

(P2) Sometimes I would go back to the mouse, but it was awkward going back and forth between the pen and the mouse.

(P2) I kept forgetting to press the button to go into navigation mode. I think it would be good if the icon changed to show that you are in navigation mode.

(P2) I also think that there is going to be greater effects when the pen and monitor are together, particularly some of the move to line stuff because it moves very quickly and because the text lines are not distinct enough visually. So that it happens so fast I can't make the association, as I'm not familiar enough with the text.

(S1) (pen broke – user used mouse)

(S1) There was one really long delay of 2 minutes.

(S1) It was pretty manageable

(S1) One request. I would prefer many more lines of text, smaller font size, and / or longer windows.

(I) Did you use the pen most of the time?

(S2) Yes, it was ok. It is just a matter of getting used to it. Often it was annoying when it didn't register.

(S3) It was slow (system problem).

(S3) I didn't use the page flicks as they were too hard to do physically. I thought the scroll bars were easier to use.

4. Was there anything about the system that confused you?

(P2) I thought that you would get a different pie menu when you hit the button and moused down on an annotation.

(S1) Yes, I lost a whole bunch of lines once and I don't know how it happened. I was trying to do a move I think and I lost the paragraph and I had to retype the paragraph. I lost it out of the edit window.

(I) Did you use the undo?

(S1) No I didn't. I have a tendency to use a very limited number of commands until I really learn something, so actually I didn't use all of the commands, but I was happy. with the set I was using. It was enough for me at the time. It didn't occur to me to use the undo. I just couldn't figure out what happened.

(S1) There were a couple of times when I was inserting and it deleted instead.

(S1) I have a big wish list. I would like to be able to click the mouse or whatever and actually just get a text cursor for little inserts. There were times when I wanted to insert just a couple of characters and I wasn't able to get the insert to work just right and I was getting really frustrated and I would have liked to say let's go to the keyboard; it'll be a bit more awkward but at least it will work.

(S2) The link. Link was supposed to move the two windows to the same spot. I couldn't tell what it was doing. It was doing something but I didn't think it was doing the same thing I was drawing on screen. I didn't use link after a while. How do you make the two screens be the same?

(I) Use goto command

(S2) How? If you did it in the annotation screen it was goto annotation.

(I) (show the goto command)

(S2) (points out problem when user clicked in the margin bug in program)

(S2) I scrolled / page flicked to line them up.

(S2) I had trouble with the insert at the beginning but got better with practice.

5. Was there anything about the system that was difficult to do?

(P1) Definitely the navigating; having physical problems pushing the button. The mouse was a lot easier.

(P1) Insert was hard.

(P2) Insertion recognition was not good. Often it was off by a character.

(P2) At one point I just wanted to make a note and I couldn't.

(S1) Inserts were rough. To insert to add just a letter, to add just a suffix, it was really hard to add a letter as it thought I was trying to insert a word. I could get into the habit of putting spaces, for example if I wanted to insert a word I could type space word space. From my editing experience, it's important to be able to modify words rather than replace them.

(S3) Changing the words was a pain, because you couldn't just change a word; you had to insert and delete. A solution would be to circle a piece of text to change. For example, use the broken move.

6. Was there anything about the system that you really liked?

(P2) I did like the ability to incorporate annotations. rather than doing it manually.

(P2) I like the split view idea; because a lot of times you're not only making annotation changes. For example, if you're annotating a paragraph which has multiple changes in it, once you make the first change you're all off synchronized so you have to kind of figure out all the next changes. That's really annoying. It's also good to have a log of what annotations were done by a particular person. so that you can go back and say this conflicts or this is the reason why I made this change because this wanted me to make this change.

(P2) Traditional tools have no way of saying why did you make these changes. Whereas with this tool you'd be able to go back and say he made these changes and he didn't take these changes.

(P2) It's weird, when I was doing the study I was actually kind of getting into it and found that it could really be useful.

(P2) The other thing, the broken moves; because a lot of times you want to be able to switch between the source and destination, say, to quickly see where does this person want to make this change and why does he want this there. So I think that's bonus.

(P2) If I have a few annotation marks in this paragraph and then there's another annotation which says I want to move this annotation to the end of the document, if I issued that last annotation will these other annotations move with that paragraph?

(I) Yes.

(S1) I like the idea.

(S2) I liked the idea. it was neat. Also when you delete something it's a lot easier to slash it out than to highlight it and then delete it. Also I liked the move.

(S3) I thought it was fun circling words.

(I) Did you try circling anything bigger like paragraphs?

(S3) No I didn't try that.

(S3) Crossing out long lines was hard because I tended to go off the words and then it would say that it didn't get it – it wouldn't recognize it.

7. Was there anything about the system that you really did not like?

(P2) If you could get more text on the screen at the same time. It's not a real big problem, but it would be nice.

8. What situations do you think a system such as MATE would benefit, as an incorporation tool for marked-up documents, as an annotation tool, or as a text editor?

(S1) In the context given.

(S1) If some of the difficulties in making small changes were eliminated, I can see it being useful in passing a document between people and if the minor problems were fixed as a text editor.

(S1) Really useful in desktop publishing.

(S2) Me, being a student of course, it would be neat if you could type in your essay and then put it in a file for your professor to take a look at it, and do all this to it, and then you can look at it yourself. Because it's a lot easier, obviously, to do it all on the computer.

(S2) I like doing my essays on the computer;. I find it very awkward printing it out. Instead of having loads of sheets of paper you have it all on a file.

(S3) I think its probably good for the end of the line in a writing process and it's a choice of either making the change or ignoring it.

9. Is there anything else which you would like to comment on?

(S1) I held both the pen and mouse at the same time.