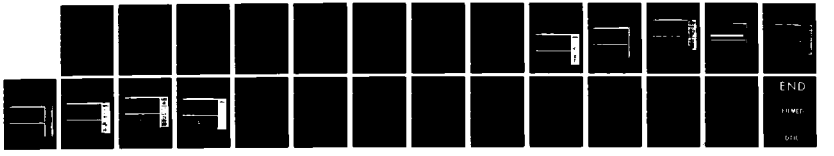
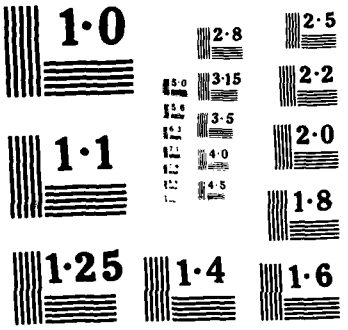
AD-A158 155 CREF: AN EDITING FACILITY FOR MANAGING STRUCTURED TEXT 1/1
(U) MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL
INTELLIGENCE LAB K M PITMAN FEB 85 AI-M-829
UNCLASSIFIED N00014-80-C-0505 F/G 5/8 NL





(2)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AIM 829	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): CREF: An Editing Facility for Managing Structured Text	5. TYPE OF REPORT & PERIOD COVERED
6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s): Kent M. Pitman	8. CONTRACT OR GRANT NUMBER(s): N00014-80-C-0505
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Artificial Intelligence Lab. 545 Technology Sq. Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209	12. REPORT DATE: February, 1985
	13. NUMBER OF PAGES: 23
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Office of Naval Research Information Systems Arlington, VA 22217	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Browsing	Knowledge Engineering	Structured Text
Document Preparation	Mail Reading	Text Editing
Editing Enviorments	Non-Linear Text	
Information Management	Protocol Parsing	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper reorts work in progress on an experimental text editor called CREF, the Cross Referenced Editing Facility.

CREF deals with chunks of text, called segments,which may have associated features such as keywords or various kinds of links to other segments. Text in CREF is organized into linear collections for normal browsing. The use of summary and cross-reference links in CREF allows the imposition of an auxillary network structure upon the text which can be useful for "zooming in and out" or "non-local transitions."

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Massachusetts Institute of Technology
Artificial Intelligence Laboratory

A.I. Memo No. 829 February, 1985

CREF: An Editing Facility for Managing Structured Text

Kent M. Pitman

MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139

Human Cognition Research Laboratory
The Open University
Milton Keynes, England MK7 6AA

Abstract

This paper reports work in progress on an experimental text editor called **CREF**, the Cross Referenced Editing Facility.

CREF deals with chunks of text, called segments, which may have associated features such as keywords or various kinds of links to other segments. Text in **CREF** is organized into linear collections for normal browsing. The use of summary and cross-reference links in **CREF** allows the imposition of an auxiliary network structure upon the text which can be useful for "zooming in and out" or "non-local transitions."

Although it was designed as a tool for use in complex protocol analysis by a "Knowledge Engineer's Assistant," **CREF** has many interesting features which should make it suitable for a wide variety of applications, including browsing, program editing, document preparation, and mail reading.

Keywords: Browsing, Document Preparation, Editing Environments, Information Management, Knowledge Engineering, Mail Reading, Non-Linear Text, Protocol Parsing, Structured Text, Text Editing.

This paper describes work done at the Human Cognition Research Laboratory of the Open University (Milton Keynes, England) and the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology (Cambridge, MA, USA). Support for work at the Open University was provided in part by Alvey Research Grant IKBS/034 and SERC GR/C/9918.1. Support for work at the MIT AI Lab was provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-80-C-0505, in part by National Science Foundation grants MCS-7912179 and MCS-8117633, and in part by the IBM Corporation.

The views and conclusions contained in this document are those of the author, and should not be interpreted as representing the policies, expressed or implied, of any of the aforementioned organizations.

This document is also available as Technical Report 14 from the Human Cognition Research Laboratory of the Open University, Milton Keynes, MK7 6AA, England.

I. Introduction

In this paper, we present the origins and current status of **CREF**, a Cross Referenced Ed ting Facility for *managing structured text.*

We begin with a look at the **KEA** (Knowledge Engineer's Assistant), which motivated **CREF**'s original design. The key ideas behind that design are presented, followed by a description of a running prototype of **CREF**. In order to put things in perspective, we compare **CREF**'s design to that of other editing facilities like it. We then close with a critical look at some of **CREF**'s shortcomings and some suggestions for future work.

It is important to understand that the purpose of this document is not to assert the correctness of some particular set of details: rather, it is to document what has been done in an experimental implementation, and to inspire interest in the ongoing development of **CREF** and systems like it.

II. Historical Perspective

This section provides some background about the Knowledge Engineer's Assistant [KEA 84], which gave **CREF** its original *raison d'etre.* This background will serve to motivate the major aspects of **CREF**'s design.

The Knowledge Engineer's Assistant

The KEA project began with a program called **TRACKER**, developed by Pollard and Church at British Telecom (BT). The **TRACKER** system, which runs in **PROLOG** [Kowalski 83] on micro-computers, provides online assistance to people doing hardware fault diagnosis of circuit boards.

The methodology used by the BT researchers to identify rules which might be of use by **TRACKER** is to study interviews with someone who is an expert at repairing a particular kind of board. The expert is allowed to speak in a relatively unstructured manner about whatever topics seem relevant to him. The conversation is recorded and later transcribed. Working under the hypothesis that such experts primarily use pre-compiled algorithms rather than general problem solving in their day-to-day repairs, the BT researchers then study the resulting transcripts and attempt to construct **PROLOG** rule-bases which capture the expert's understanding of his own behavior.

On two occassions, transcripts were studied without mechanical aid and on both occassions, the BT researchers have succeeded in obtaining a satsifactory set of **PROLOG** rules. However, in both cases the process was painstakingly slow. It required several months of leafing back and forth between pages, developing indexes, underlining and highlighting, performing searches, making notes in margins, drawing diagrams, and writing code — all of this done primarily by hand.

The goal of the **KEA** project is to identify which aspects of this task can be mechanized, and to implement such mechanization.

Design Criteria for CREF

Two important capabilities which have been identified as essential to a **KEA** are the **ability to support protocol analysis** (or perhaps knowledge elicitation in general) and the **ability to represent and manipulate theories.**

The **CREF** system provides a first approximation to a **KEA** by providing tools for editing, annotating, and cross-referencing information. Even in its current state, it provides a great deal of leverage in the area of protocol analysis. It also provides some support which relates to the creation and manipulation of theories about task domains, but this aspect is not as well fleshed out and should be a focus of attention in later **KEA** development efforts.

CREF itself is not a knowledge-based utility, but it does provide structures and functionality which should prove to be an appropriate foundation for further work on the knowledge-based aspects of the **KEA** project. The methodology applied in **CREF**'s design was to study what the BT researchers had

been doing by hand and then to assure that CREF could provide formal analogs of the structures and actions they were accustomed to using.

This approach was important for several reasons. It meant the researchers (some of whom were not computer users) would not feel like they were being coerced into using some new methodology. They could continue to apply techniques with which they were familiar, usually relying on existing intuitions about what needed to be done and in what order. The primary effect of having the machine present was to speed up those techniques.

Also, this approach will make it possible to provide a control situation for productivity measurements after the later introduction of knowledge-based tools, since any speed-up due to simply having moved to a mechanized approach could be measured in this initial system and usefully taken into account. If knowledge-based tools were added at the same time as the initial mechanization of rote tasks was introduced, then it would be hard to tell whether any apparent speed-up caused by the automated system came from its knowledge-based nature or simply from the fact that machines are much faster and more reliable at many mechanical tasks.

Empirical Basis for CREF

The rest of this section itemizes the techniques the BT researchers were observed to use during their work prior to **CREF**.

Editing/Revision. A pre-pass was typically made over initial protocols (transcriptions of conversations with experts) to remove irrelevant text that did not relate to the domain being studied. A second document was then produced which was slightly smaller and less prone to be interrupted by irrelevancies. No further revisions were likely to be made, however, because of the tedious nature of copying the document and the fact that after that point there was a significant build-up of annotation and cross-reference information which might be lost in such a revision.

One researcher noted that she never threw out any notes that she ever made because she never knew *what she might want to refer back to at a later time.*

Segmentation. Protocols were divided into numbered segments so that cross-reference could be done at a finer grain than per-page. Segment boundaries were drawn each time speakers alternated in the conversation (approximately every paragraph or two).

Indexing. Additional pages were added to the protocols containing key words or phrases and numbers corresponding to segments where information about these keys could be found.

Highlighting. Highlighting and underlining were used to annotate key ideas within segments. In the case of highlighting, the color was sometimes varied in order to convey additional information.

Margin Notes. Notes were written in the margins, usually clarifying wording or noting how several contiguous segments were related.

Pictures. Many kinds of pictures were created during repeated passes over the text in an attempt to form models of relations alluded to by the text (*e.g.*, class hierarchies, part hierarchies, causal chains, decision flow charts, physical descriptions of objects, and logical descriptions of objects).

In the next section, we will see how each of these items had direct influence upon the design of **CREF**.

III. Overview of CREF

This section describes the essential concepts and structures which make **CREF** interesting. Some of these features may change in later designs, but it is important to define clearly what exists now so that it may be studied critically later.

Segments

A **segment** is a block of text which is treated as a unit by **CREF**. Local editing operations are possible within a segment, but a segment is atomic in the sense that most **CREF** operations work at the level of segments or more complex structures; rarely is the textual part of a segment examined by the **CREF** system.

The choice of segment size is arbitrary; a segment may contain a word, a sentence, a paragraph, or even many paragraphs. The decision about how to break up the text being edited into segments is left to the user and may vary widely depending upon the application. In our experimentation, however, a segment typically refers to a paragraph or two of text, a program definition, or some sort of table.

Operations are provided in **CREF** for splitting one segment into several segments and for joining several segments into one. This means that any decision about granularity found to be inappropriate for some application can be changed straightforwardly at any time.

A useful metaphor which has been used in the **CREF** system for visualizing segments is that of index cards. Each segment corresponds conceptually to an index card. Many **CREF** operations are most easily visualized by appealing to this metaphor.

Collections

A **collection** is a possibly ordered set of segments. Collections are one of several ways that **CREF** imposes structure on segments.

An **abstract collection** is described by a predicate which, given a segment, returns true if and only if the segment is part of the collection.[1] Elements of an abstract collection need not be remembered explicitly since they can always be recomputed. If some **CREF** command operates on the segments in an abstract collection, the order in which the segments in that collection are touched is not defined.

A **static collection** is defined by a membership list (which is mutable). Static collections typically begin as abstract collections, which are then modified or frozen so that their exact contents and ordering can be retrieved at a later time.[2]

In practice, collections are used in a manner similar to the way buffers or files are used in conventional text editors (*e.g.*, **EMACS** [Stallman 81]). They partition the space of available data so that it doesn't have to be worked with all at once. **CREF** always has a **selected collection**, the contents of which can be edited using essentially conventional editing techniques.

Some readers may recognize that the **CREF** notion of collection was derived from the idea of "surveys" in the **BABYL** mail reader [Ciccarelli] and "collections" in the **ZMAIL** mail reader [Handel 83]. **CREF**'s collections differ from these in that it presents all the segments belonging to a collection for editing at the same time rather than offering a menu and allowing the user to select segments one at a time. Hence, the presentation style is more visually similar to that used by **CONVERSE**, the interactive message handling facility used on the Lisp Machine.

[1] An abstract collection is similar to a "view" in conventional data base terminology [Gray 77].

[2] A static collection is similar to a "copy" in data base terminology.

Keywords

CREF allows the user to attribute one or more features to a segment by associating **keywords** with the segment.

The keyword facility provides information similar to that provided by an index in a hardcopy document. It provides the ability to select collections based on boolean combinations of keywords. Keywords have been used for a long time to annotate messages in mail reading systems (*e.g.*, **MM** [McMahon], **BABYL**, and **ZMAIL**), but have no analog in conventional text editors.

It is also possible in **CREF** to select a collection of segments which match a given search string, but such matching is often less fruitful than keyword matching. This is because users tend to associate semantic content with their keywords while search strings are necessarily purely syntactic.[3]

Relational Links between Segments

Segments in **CREF** may be annotated by **links** to other segments to express various kinds of relations. Most traditional text editors have no analog to this because they offer little or no way to treat regions of text as first class object which could be pointed to by such links.

Currently, **CREF** uses the following **link types:**

REFERENCES links are used to point to segments which may provide related information that is possibly not of general interest or which is somewhat off the subject. The way in which reference links are used can vary widely depending on the application. For example, in a document they might point to footnotes or references, but in code they might point to comments, justifications, or proofs.

SUMMARIZES links are used to impose a hierarchy upon a given set of segments. A "summary" is just a piece of text (*i.e.*, a segment) which has a summary link to one or more other segments. A segment may have more than one summary, and summaries may themselves have summaries.

SUPERSEDES links are used to implement versioning. When a user "freezes" a copy of an individual segment, a copy of it is made which he continues to edit. The original copy is kept (and is pointed to by this link) in case he wishes to return to it later. All keyword information and most of the link information are copied from the original segment, and segments which pointed to the original segment are updated to point to the new version.

PRECEDES links record information about ordering. In general, this information is not used to control presentation order. Rather, it is used in situations where the presentation order violates some other more important order. For example, when editing code with **CREF**, there may be several collections each editing the same set of functions, each with its own presentation order. Yet there may be only one ordering which will assure the correctness of the code when it is time to compile or execute it. PRECEDES links are intended to record such additional ordering information.

Diagrams

In addition to providing ways to manipulate information as text, **CREF** also provides facilities for manipulating information pictorially through the use of **diagrams**, which allow the user to record and use certain kinds of domain-specific knowledge in graphical form.

This diagram facility is integrated with **CREF**'s indexing capabilities. Using a mouse[4], the user can give commands that relate to designated areas of a diagram. For example, by pointing with the mouse, the user can ask to see a collection containing segments related to a designated region of the **selected diagram**, or he can ask to see a menu of other diagrams which might be related to the designated region.

---

[3] The problem with search strings being "syntactic" is that they sometimes match unwanted text or miss semantically related text which was not in the proper form to be matched syntactically.

[4] The mouse is the primary pointing device used by the Lisp Machine.

Domains

In an editing system, it is necessary to partition the workspace so that if a user is working on two unrelated projects at the same time, the artifacts of one project do not get mixed up with those of the other. In **CREF**, we call such partitioned workspaces **domains.**

Theories

Within a given domain, it may be interesting to instantiate different organizational patterns for some shared initial information. For example, two workers independently assigned the same transcript to analyze might want to share the transcript but any additional data or organizational structure they created might want to be kept partitioned. **CREF** allows domains to be subdivided into theories for this purpose.[5]

In general, when people speak of having a theory of something, they usually mean they have organized their thoughts about one topic using one or more formal representation schemas such as diagrams, graphs, tables, constraint relations, logical assertions, equations, or procedures.

A theory in **CREF** is similar. It is a set of collections, segments, diagrams, and keywords which work together to form an organizational structure. The theory being actively used at any given time is called the **selected theory.**

Only objects (segments, diagrams, keywords, *etc.*) which belong to the selected theory are active. For example, if the user asks to edit all of the segments in some abstract collection, the segments which will be considered for editing are only those which are in the selected theory. Some objects may be explicitly marked **inactive** even though they are part of the selected theory. This keeps them from being considered in references to abstract collections without requiring them to be flushed from the theory. For example, segments which have been superseded are typically marked inactive.

[5] Of course, if two users wanted to work together on a project, they could simply agree to work within the same theory.

IV. Using CREF

In this section, we present a few simple examples of **CREF** to help illustrate some of its key features and how they manifest themselves in practice using the current implementation of **CREF** for the Lisp Machine.

The CREF Frame

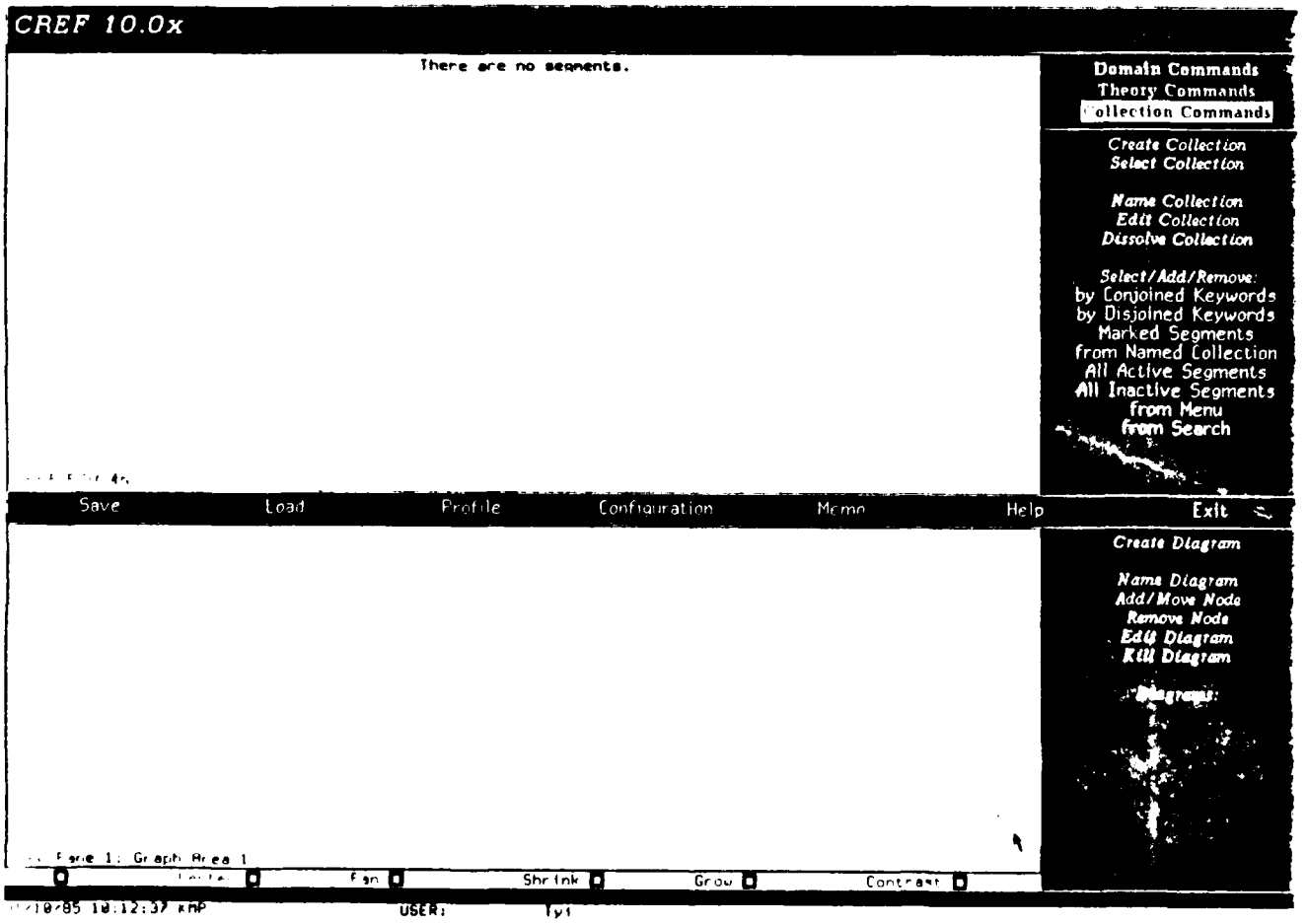
In its initial configuration, the **CREF** screen is divided into several parts.

Taking up most of the top of the screen is the **Edit Area**. In this area, collections of segments are displayed for editing.

Taking up most of the bottom of the screen is the **Graph Area**. In this area, the user can construct useful diagrams to complement the information available in the Edit Area. Along the bottom of the Graph Area is a special purpose menu for manipulating the presentation of graphs which are too large to be displayed in the Graph Area all at once.

Along the edge of each of these two main areas is a menu containing useful commands for the area. In the case of the Edit Area, there are actually three possible menus, only one of which (the one called "`Collection Commands`" in this case) is offered at any given time.

Separating the two main areas is a thin menu of useful operations which relate to **CREF** as a whole.



Getting Started

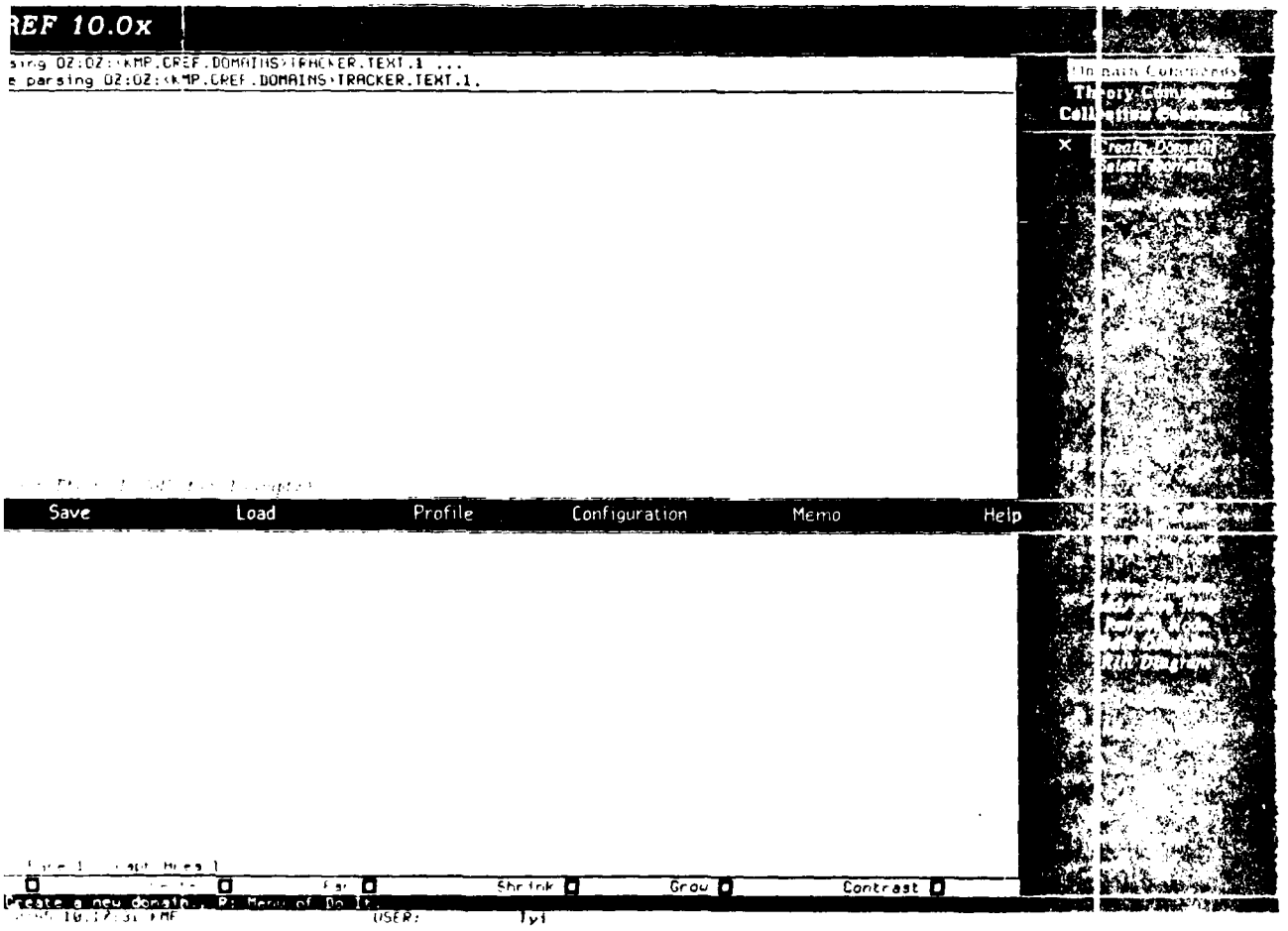
The CREF user can begin his work in three ways:

Loading a Saved Workspace. The user can re-enter a saved workspace (domain and theory) by using the `Load` command (which appears on a menu in the middle of the screen).

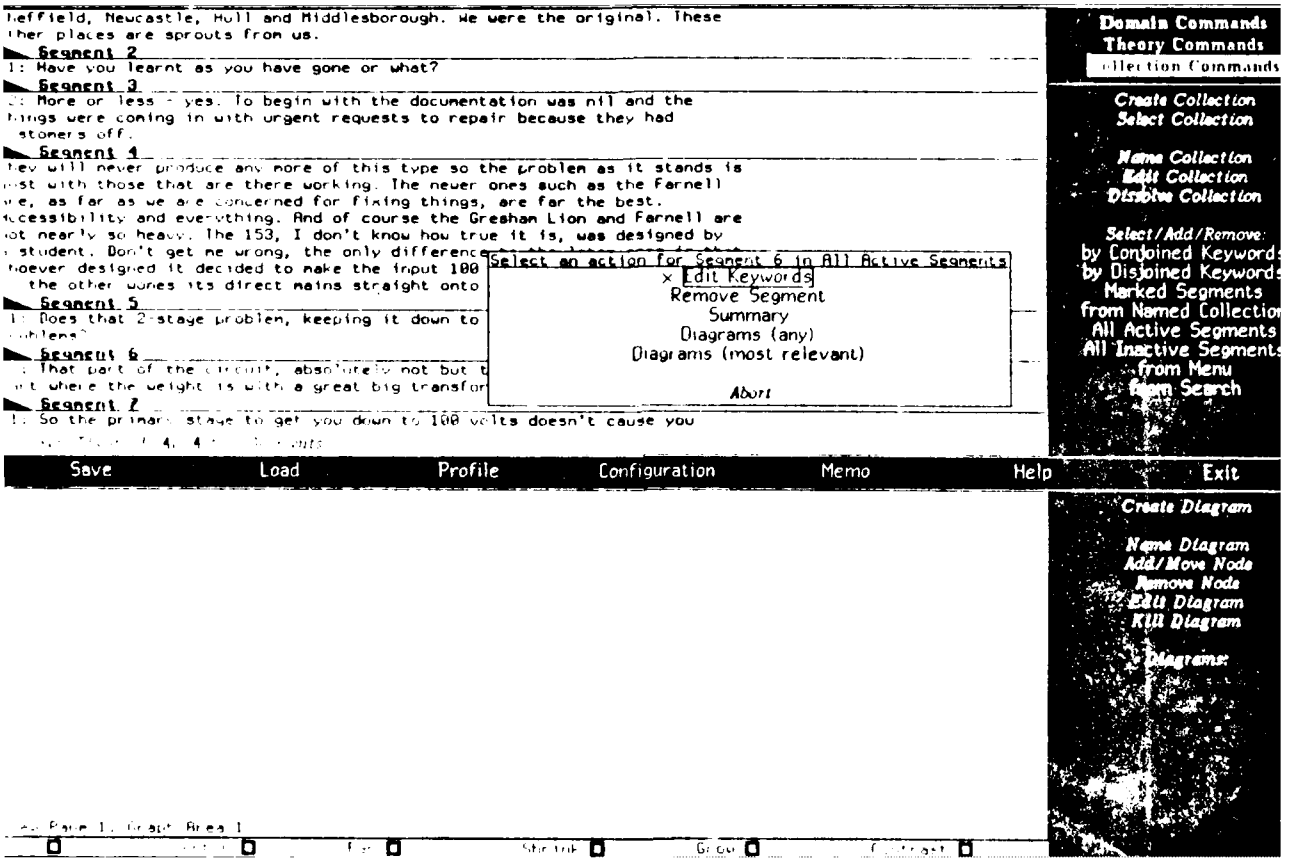
Segmenting Raw Text Files. The user can have a raw file of text (such as a transcribed protocol) broken into segments (at paragraph boundaries) and use those segments to "seed" his theory.

In the illustration below, the user has selected the "`Create Domain`" menu item from the domain commands at right. He has specified a domain name, "`Tracker`," from the keyboard. The CREF system, noticing that there is a protocol (text file) associated with that domain name has automatically segmented that file (at paragraph boundaries) so that it will be available for perusal, annotation, and revision.

Creating a New Workspace. If a protocol file or saved workspace does not already exist, the user can enter an empty domain (or a new theory within some domain) and build up the complete contents of the domain (or theory) from the keyboard. By typing `Hyper-I`[6], he can create a new segment and have it placed in the collection he is editing.



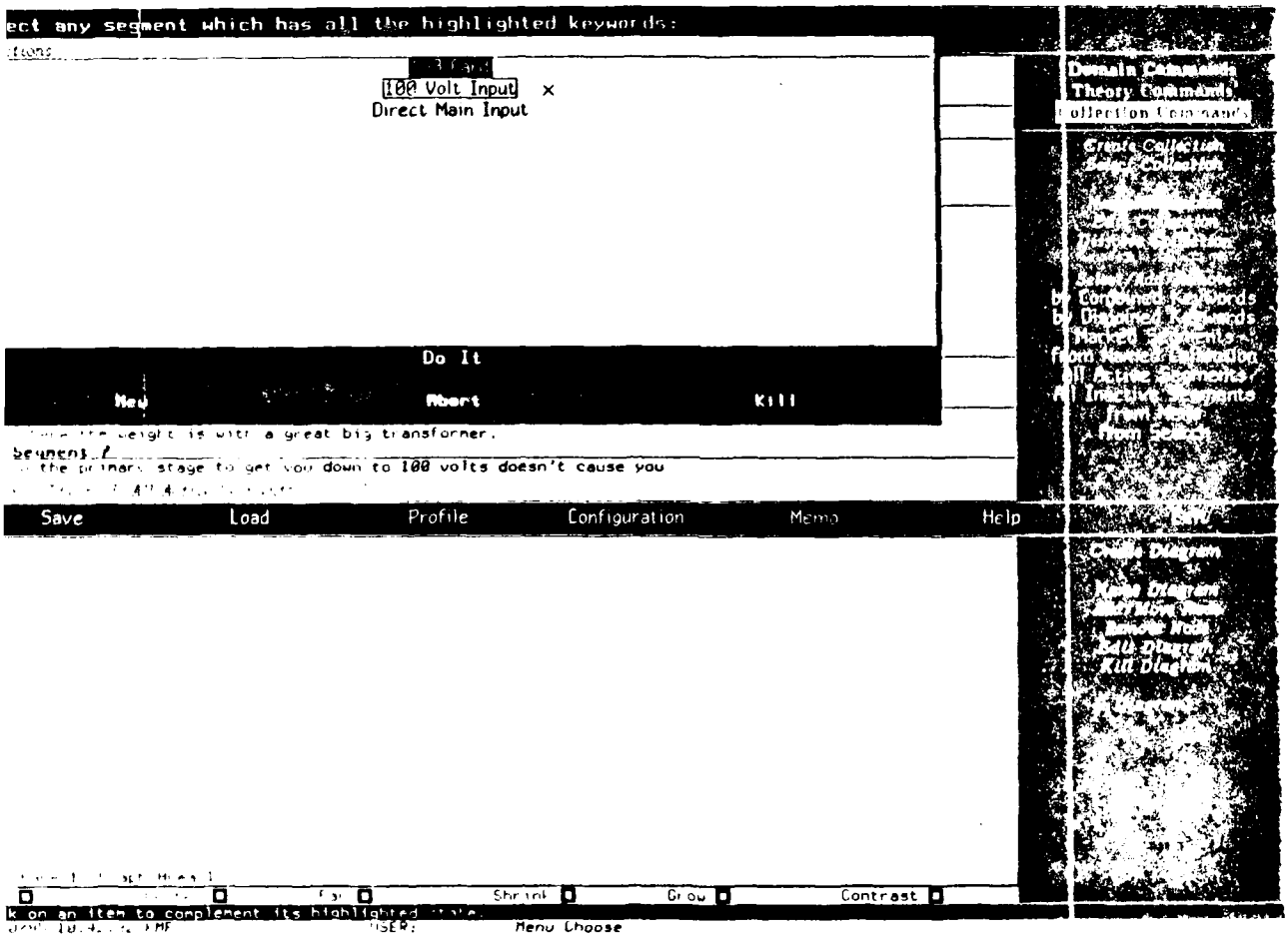
[6] In addition to the conventional `Shift` and `Control` keys seen on most modern computer keyboards, Lisp Machines offer a number of additional shift keys, including `Meta`, `Super`, and `Hyper`, which can be used separately or together to modify keystrokes. Hence, `Control-X`, `Super-Y`, `Hyper-Control-Z`, and `Hyper-Super-Meta-Control-X` are examples of possible "keystrokes" (sometimes called "chords," by analogy to the playing of a musical instrument).


heffield, Newcastle, Hull and Middlesborough. We were the original). These
ther places are sprouts from us.
Segment 2
1: Have you learnt as you have gone or what?
Segment 3
2: More or less - yes. To begin with the documentation was nil and the
kings were coming in with urgent requests to repair because they had
stomers off.
Segment 4
hey will never produce any more of this type so the problem as it stands is
ost with those that are there working. The newer ones such as the Farnell
re, as far as we are concerned for fixing things, are far the best.
ccessibility and everything. And of course the Gresham Lion and Farnell are
ot nearly so heavy. The 153, I don't know how true it is, was designed by
student. Don't get me wrong, the only differenc
hoever designed it decided to make the input 100
the other wones its direct mains straight onto
Segment 5
1: Does that 2-stage problem, keeping it down to
oblems?
Segment 6
: That part of the circuit, absolutely not but t
it where the weight is with a great big transfor
Segment 7
1: So the primary stage to get you down to 100 volts doesn't cause you
Select an action for Segment 6 in All Active Segments
× Edit Keywords
Remove Segment
Summary
Diagrams (any)
Diagrams (most relevant)
Abort
Domain Commands
Theory Commands
Collection Commands
Create Collection
Select Collection
Name Collection
Edit Collection
Dissolve Collection
Select/Add/Remove:
by Conjoined Keywords
by Disjoined Keywords
Marked Segments
from Named Collection
All Active Segments
All Inactive Segments
from Menu
from Search
Save
Load
Profile
Configuration
Memo
Help
Exit
Create Diagram
Name Diagram
Add/Move Node
Remove Node
Edit Diagram
Kill Diagram
Diagrams:

Selecting Collections

Once keywords have been attached to some or all of the segments in a given theory, it s possible to select a collection of segments having (or not having) some given keyword(s).

In the illustration below, the user has clicked **Mouse-Left-1** on "**by Conjoined Keywords**," which requests that **CREF** select a collection containing only segments which have all of a given set of keywords. The set of keywords to use for this selection is being chosen from a menu of all keywords active in the current context (domain and theory).

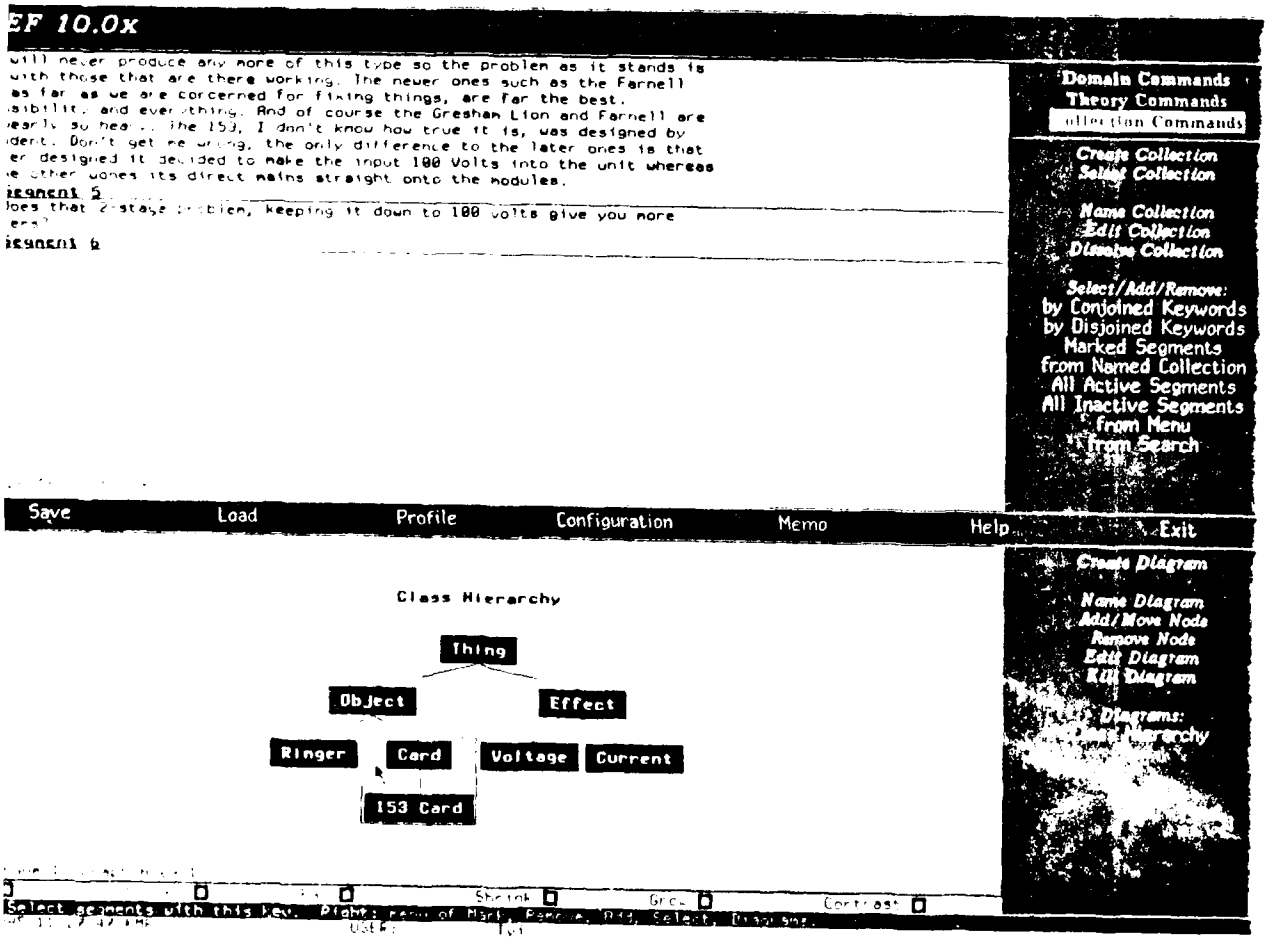


Having selected a collection in this way, it is possible to change its contents. For example, if the user has selected segments with keywords "**153 Card**" and "**100 Volt Input**" and later discovers that he would like to see a slightly different set of segments instead, he can refine the collection's contents using the "**Add ...**" and "**Remove ...**" options on the menu at right. For example by clicking **Mouse-Middle-1** on "**from Search**" and giving the string "**153**" he could add all segments which have the string "**153**" anywhere in them.

Diagrams as a Selection Device

In addition to their obvious use as an organizational tool for helping the user think about the relationships between concepts in his problem domain, diagrams serve a very important function as an indexing device within **CREF**.

The user may, by simply pointing at a region of a diagram, designate which segments he would like selected. In the case below, the user has pointed at "`Card`" (and its inferiors), so CREF has selected a collection containing exclusively segments which had either the "`Card`" keyword or the "`153 Card`" keyword.



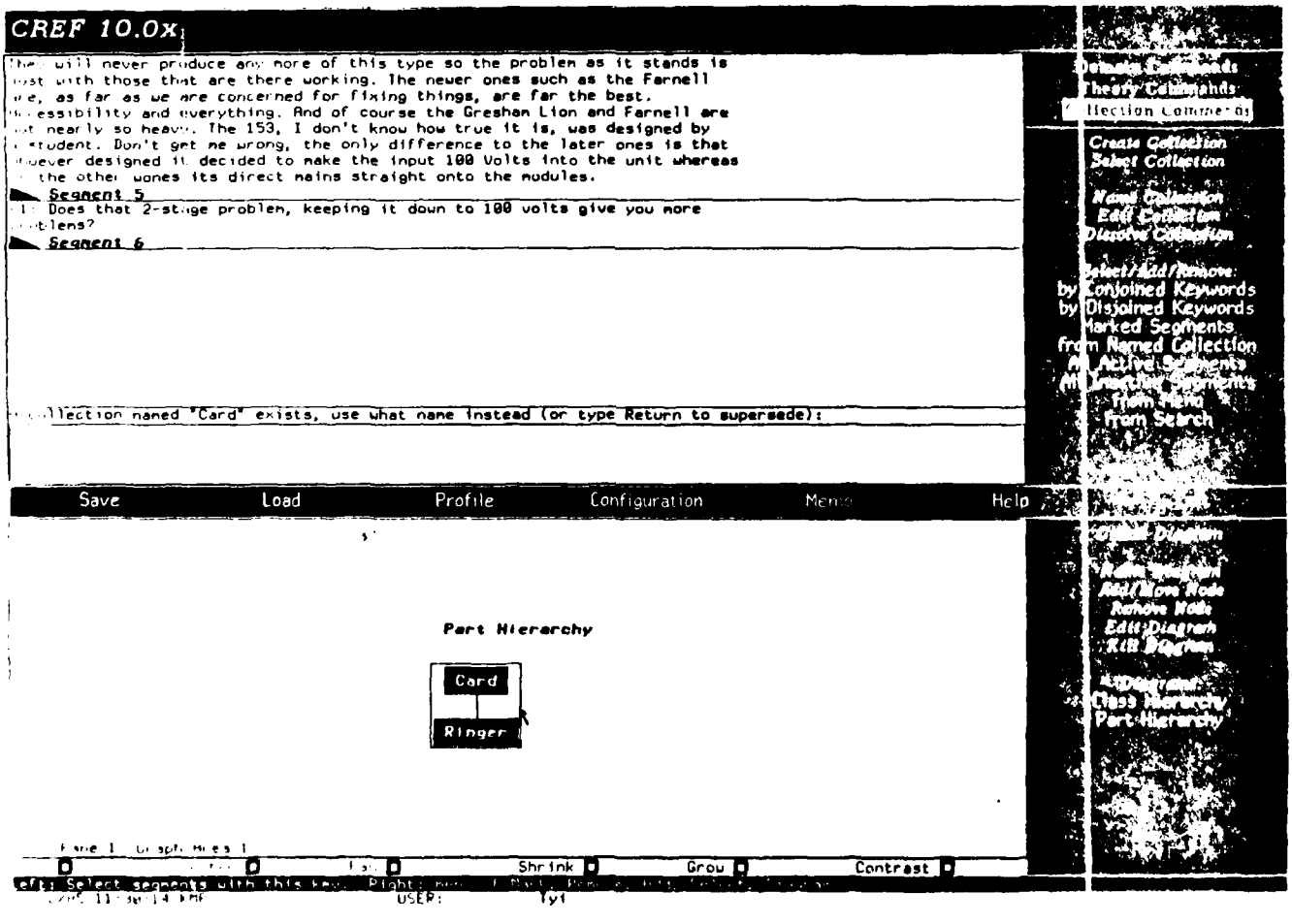
An important advantage of this mechanism is that the user need not explicitly label things "`Card`" in order to be able to reference them as such later. He can label things as "`153 Card`" if they are only specifically about that kind of card and then independently assert (through the diagram facility) the information that a 153 card is a kind of card.

Diagrams as Points of View

It might happen, in fact, that there are competing views on how keyword implication could work. For example, the keyword "Card" might seem to imply one thing in the context of the class hierarchy on the previous page and yet another thing in the terms of the part hierarchy shown below.

CREF always uses the selected diagram to provide the appropriate advice about keyword implication. In the example below, the user has again asked to select a collection based on the "Card" keyword, but this time the set of inferiors are not the same as they were when last that collection was selected. The system has detected a conflict and informs the user by asking the user what name to use (instead of "Card") for the specified collection.

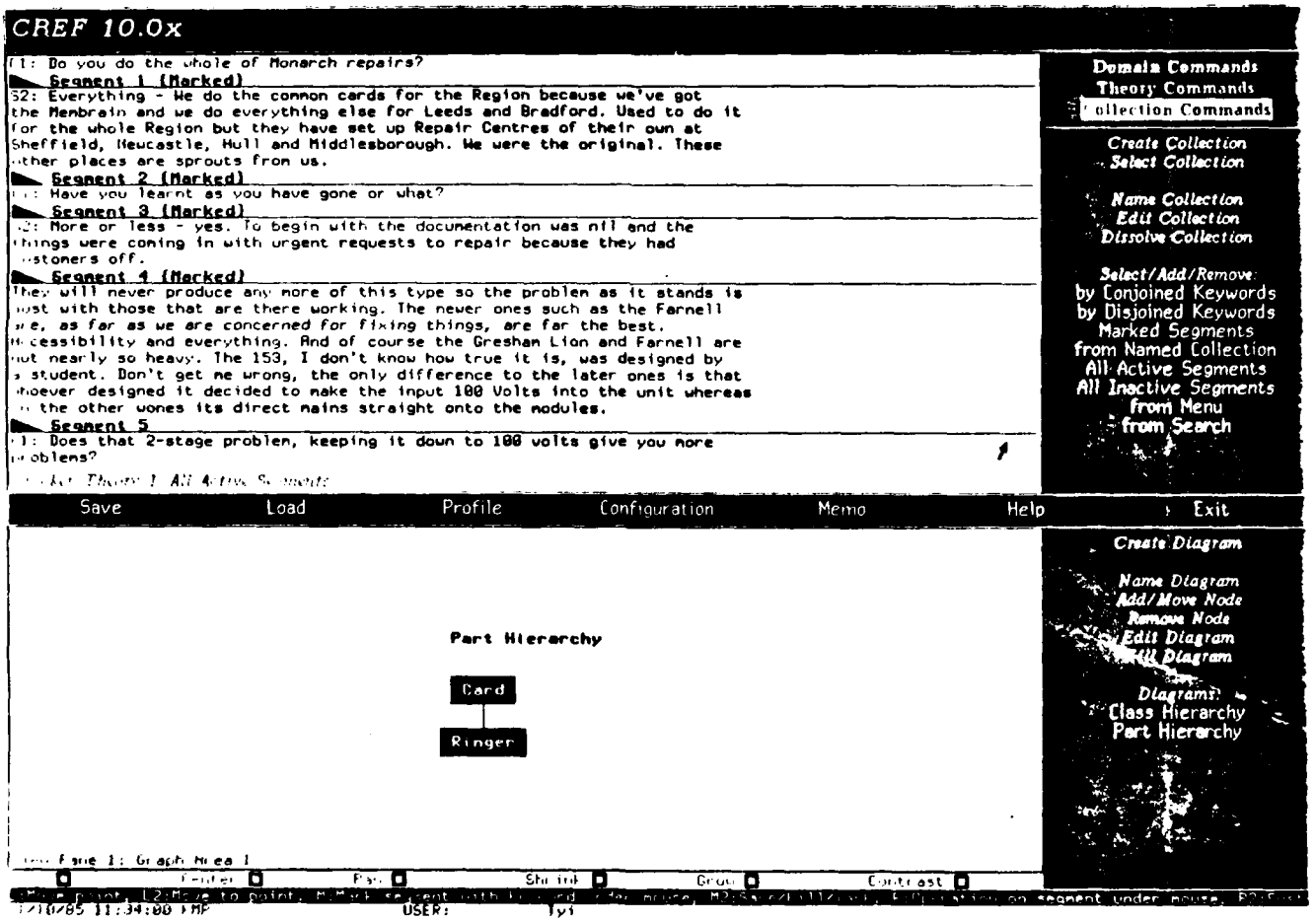
The implication mechanism used in these selection examples is quite general, allowing each kind of diagram to define its own implication behavior. As CREF considers each segment for inclusion in the set of things to be selected, it uses a message passing protocol to simply tell the diagram what keywords are explicitly attributed to a given segment and ask whether the given keywords imply the desired keyword. For example, an "and/or tree" might behave very differently than a "hierarchy" with regard to implication.



Commands That Manipulate More Than One Segment

Sometimes there is an operation we would like to do to several segments in a collection. For example, the join command, `Hyper-J`, will replace several segments with a single segment containing the joined text of all those segments. Another command, `Hyper-S`, allows the user to create a segment which summarizes a designated set of segments so that he can go back and forth between short and long versions of the text in those segments.

In order for the commands like these to know which segments they are operating on, there are mechanisms for pointing at a number of segments at the same time. One such way is to use the command `Hyper-Control-Space`[8] to individually mark each relevant segment as being an argument to the next command. In the illustration below, `Segment 1` through `Segment 4` have been marked using this command.

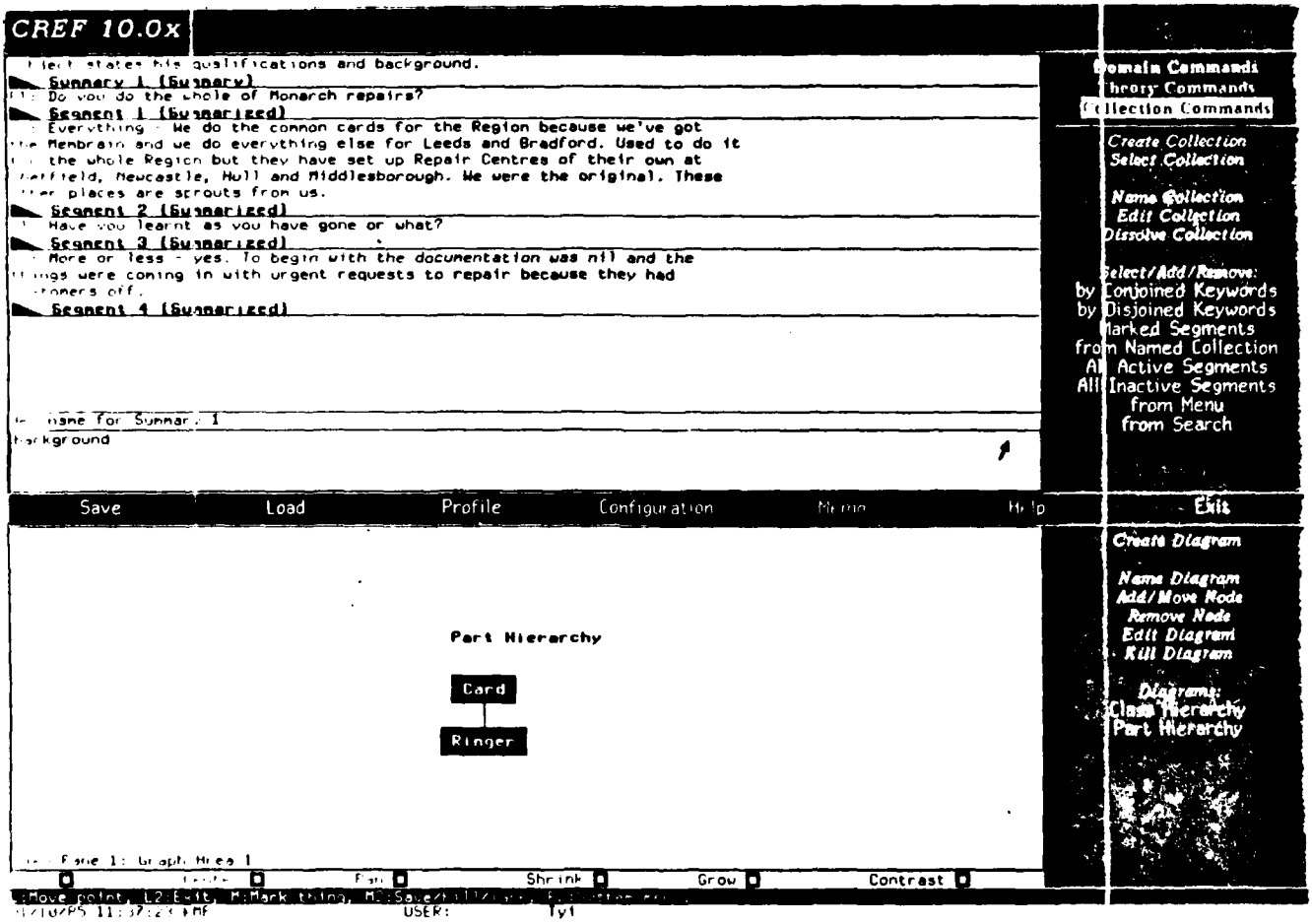


[8]This name would be mnemonic to users of ZMACS or EMACS since it is analagous to `Control-Space`, which also has the effect of "making a mark."

Creating a Summary

If the user types **Hyper-S**, he is placed in a collection containing one new segment and all of the segments he has just marked. The new segment, which has a generated name such as "**Summary 1**" is empty and he must initialize it with some text that summarizes the contents of the other segments.

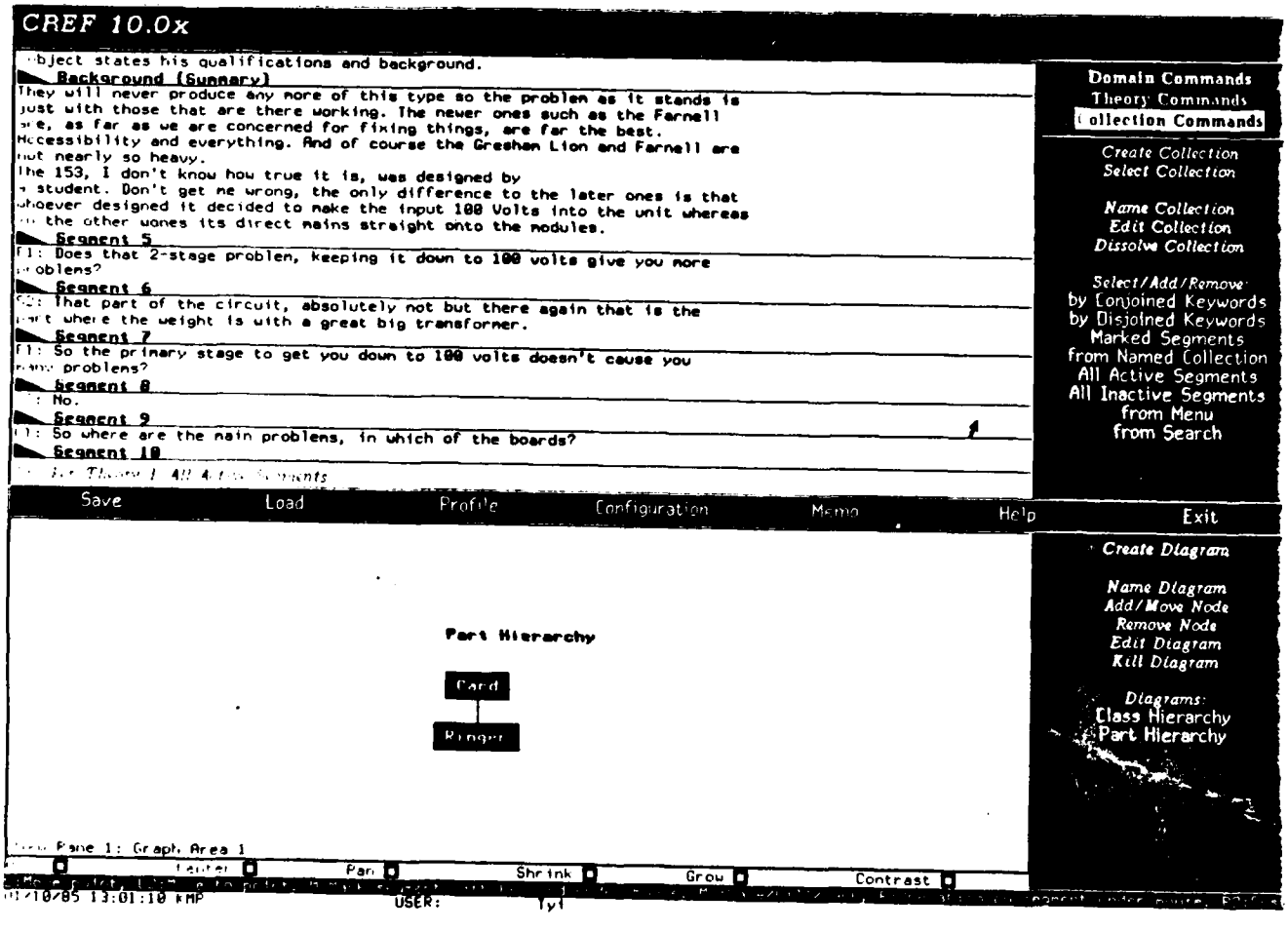
In the example shown below, the user has given a very short summary of the marked nodes and has typed **Hyper-N** to name the current summary "**Background**" instead of "**Summary 1.**"



Summarizing and Expanding

Having created a summary, the user can return to the previous collection where (as illustrated below) he will see that the segments he had marked have been replaced by the summary he just created.

At any time, if he wishes more detail, he may place his editing cursor within the summary segment and type `Hyper-D` to expand the summary (replacing it with the nodes it summarizes) and `Hyper-U` to contract the summary back. The "D" and "U" being mnemonic for a sense of "down" and "up" in the hierarchy that summaries impose upon segments.



V. Applications and Related Work

In this section, we will survey a number of text editing applications, comparing **CREF**'s functionality with that of other interesting programs that address the same issues.

Text Editing

Other editing systems address the same issues as **CREF** addresses, although the specific mechanisms provided by such systems vary widely. Most notable among these "alternate" systems are **NLS** [Engelbart 68,70], **TEXTNET** [Trigg 84], and **BOXER** [diSessa 84].

NLS allows text to be arranged in a hierarchy and then provides tools for viewing and manipulating that structure. An interesting feature of **NLS** which is neglected in conventional text editors is that of depth abbreviation. It is possible to conceptually "zoom in and out" of text by writing viewspecs which specify that a certain presentation should ignore various levels of detail. **CREF** provides a similar feature in a different way by offering commands for the creation of segments which summarize others and commands which replace segments with their summaries and vice versa.

TEXTNET also has a structured model of text. It uses two kinds of nodes, "chunks" (text nodes) and "tocs" (tables of contents, which have no text part). Chunks correspond to segments in **CREF**, but tocs have no direct analog. A user of **CREF** would just use a collection of summary nodes to represent an outline or a table of contents; there is no special purpose data structure explicitly for doing indexing. **TEXTNET** provides for three basic kinds of movement through text: "horizontal," "vertical," and "hypertext." The approximate counterparts to these in **CREF** are scrolling through collections, expanding and contracting summaries, and tracing cross-reference links, respectively.

BOXER provides not only a structured model of text, but a complete programming philosophy to match it. It is much less like **CREF** than **NLS** or **TEXTNET**, but has some interesting features in common. **BOXER** introduces the notion of a "box" of data as a sort of universal container able to contain other boxes or more primitive data such as text. Both **BOXER** and **CREF** suggest the need to move away from linear files with no internal structure in favor of more complex structures which can "point into" one another.

Rather than linearizing text for presentation, as is done in **CREF**, **NLS** and **TEXTNET**, the display of **BOXER** data is may be abbreviated in two different dimensions according to the structure imposed by the boxes. Although its strategy for presentational abbreviation is much different than that used by these other systems, the goal is the same: the screen can be made to contain a higher density of useful information because the abbreviation is done by filtering out levels of detail rather than simply performing length truncation, which is the only method of abbreviation available in most conventional text editors.

Browsing

With its ability to peruse linear text, expand or contract summaries, and trace cross-reference links, **CREF** is an ideal tool for document browsing. In fact, its original purpose was to provide knowledge engineers with the ability to impose structure on text for the purpose of browsing.

Many of **CREF**'s browsing features were inspired by the **INFO** program [Stallman]. From inside **INFO**, it is possible to access the full mechanism of **EMACS**, but **INFO** itself does not try to take the place of a general purpose editor. Specifically, **INFO** allows users to view a document as if it were a directed graph by dividing up files into smaller units which can be addressed by name for the purpose of jumps. The idea for SUMMARIZES and REFERENCES links in **CREF** was suggested partly by experience with menus and footnotes in the **INFO** system.

Mail Reading

In most contemporary computer systems, separate programs are provided for reading/sending mail and for editing text. If a piece of of code or a correction to a document is received in mail, it typically cannot be directly accessed from the editing system. The normal procedure in such a case is to copy the message text to a temporarily location (such as a file or a "kill ring"), move to the text editor and yank the text back. Most text editors do not provide for files which "point into" other files.

Since most of the useful features provided by mailreaders such as **MM**, **BABYL** and **ZMAIL** have been incorporated into **CREF**, there is no longer a good reason to separate the notion of "mail reading" from general "text editing." The normal text editing data structures in **CREF** lend themselves so directly to mail reading that a separate program for mail reading should not be necessary.[9] In addition, segments received in mail could usefully contain code or commentary about code without having to be moved from the mail file.

Editing Code

The encouraged style for editing code in the Lisp Machine's **ZMACS** editor is relatively non-linear in character, though the structure of the text itself is strictly linear. There is a command which allows the user to go directly to the definition of any function without the user having to name the buffer in which the definition resides. This is especially important in situations such as those arising with the **ZETALISP** flavor system (or any linguistic constructs that allow for definitional inheritance) because the user may not know where an inherited method definition was defined.

ZWEI, the underlying subroutine base upon which **ZMACS** is built, offers the capability of imposing a hierarchical node structure on a set of lines in a buffer, but the capability is somewhat limited. The relations it allows are restricted to be strict hierarchies rather than general directed graphs. Also, because the node structure is not intended to be explicitly manipulated by the user, there is no visible marker which visually separates the nodes and hence no reliable way for the user to detect which node he is modifying. Since the **CREF** user's sense of focus is controlled by which set of segments is selected at any given time, it is important that he be able to tell which segment is being modified at any given time.

By appropriate manipulation of collections, **CREF** users can not only go directly to the text for a particular definition (as they could in **ZMACS**), but can also view definitions in a variety of different arrangements. For example, Lisp Machine users have from time to time complained that the linear nature of files is irksome when editing flavor definitions because sometimes they want to see all the method definitions of a certain flavor together, but other times they want to see all of the definitions for a certain method name together (regardless of the "owning" flavor). In practice, users of conventional text editors must decide on exactly one presentation for their programs, but **CREF** could allow multiple, orthogonal presentations through its collection mechanism.

Another problem with unstructured text files is that comments are not separated from code. Since the common convention in contemporary systems is to determine whether a file needs recompilation by noting when it was last modified, programmers are discouraged from editing comments in files which are heavily depended upon by other files since such editing may later force large amounts of unnecessary recompilation. An editor like **CREF**, which has enough structure to provide a natural partition between comments and code, could retain per-segment modification information so that a change to a comment would not force recompilation.

Also, because the comment is not part of the code, no special characters would be required to identify it as a comment. In conventional text editors, the normal commands for justifying text or indenting code frequently do not work on comments because those commands may be confused by the column of semicolons down the left margin. Since the status of a piece of text as a comment in an editor like **CREF** is not a result of the text making it up, the text of a comment could take on any desired shape and the normal commands for justification and indentation require no special understanding of comments in order to work correctly. In fact, since even the command to create a comment for a segment would work in a comment, meta-level comments might become quite common.

[9] The designers of the **BOXER** system make a similar claim.

Document Preparation

Another interesting area to explore would be general text editing and document preparation. Since this activity requires frequent reorganization of text in a structured fashion, CREF's ability to adjust its 'focus,' manipulate logical chunks as a unit, and to maintain multiple presentation orders could probably be quite useful.

However, as the author discovered during an abortive attempt to produce this paper using CREF, the writing of papers is a much trickier business than the writing of machine-readable documentation or code. While CREF provides many gross editing operations that are of use in paper writing, more work on the finer areas of layout (*e.g.*, font control) would considerably enhance its usefulness in document preparation.

Interfaces

One way to establish an interface between an application program and the user is to extend the normal text editor interface so that normal editor commands can act as a communication channel between the user and the system being developed.

For example, in the Programmer's Apprentice (PA) [Pitman 83], the user may request that certain knowledge-based editing commands be performed upon the program definition which he is currently editing. The "apprentice" (a software module) must then parse the buffer to determine what region is being referred to, act upon the relevant data, and perhaps add some new definitions or change some existing definitions in the buffer.

This sort of facility typically involves a great deal of special purpose string manipulation (for detecting what parts of the buffer being edited are relevant to the current command). Though not an interesting part of the interface, it is a necessary part because the granularity at which the editor deals with data (buffers) and the granularity at which the PA deals with data (program definitions, comments about definitions) are not the same.

Interfacing expert systems like the PA to CREF would be considerably easier than interfacing them to ZMACS because the interface could work at the level of segments, in many cases ignoring the textual structure. For example, rather than parse the buffer to find where a comment for a function might be inserted, the PA could just create a REFERENCES link from the segment for a definition to a segment the PA had created for the comment. Also, if the PA wanted to modify the definition of a function, it would supersede the previous definition (using a SUPERSEDES relation) at the abstraction level of segments rather than at the abstraction level of unstructured text.

VI. Directions for Future Work

In using CREF, a number of issues have been raised which are not adequately solved in the current implementation. In this section we survey those issues on the assumption that information about CREF's weak points will be as important as information about its strong points to designers of future systems like CREF.

Command Set

CREF has a rich connective structure and many useful commands for manipulating that structure. Since it has so far had no "real users," its command set is certainly not optimal. Careful study must be done as it evolves to verify that all operations which are logically possible and useful given the available structure are possible to perform, and that commonly needed operations have a pleasant interface. The current command set is optimized for manipulation of already-written protocols and may need more work in order to "feel good" for other applications.

Presentation Issues

Segments are currently presented in a "rectangular" style. That is, each segment is displayed as some integral number of lines of text. No line of text is ever shared between two segments. Each segment is followed by a **banner line** which identifies the segment by name and which may contain useful information about the segment (such as an indication of the fact that it is a summary or that cross-references are available). Banner lines also provide the useful function of visually separating segments from one another so that segment boundaries are always clear.

The banner lines separating segments occasionally seem visually intrusive and always cost vertical space on the screen. At a later time, it would probably be worth experimenting with alternate presentation styles, including those which involve modified ideas about the nature of segmentation (such as allowing segments to be non-rectangular or permitting segments to overlap and share text).

Diagrams vs Segments

There is a distinction in **CREF** between diagrams and other segments. Text is broken into segments and placed in collections. Diagrams are handled in a separate window, independently of segments. An examination of the notes made by the BT researchers in their work shows that diagrams may occur interspersed with text.

In looking back at **CREF**'s design, it seems clear that the design of **ZWEI**, which contains only very limited facilities for mixing graphics (and "mouse sensitivity") with text had been a substantial influence on the design process which led us away from thinking of diagrams as "just another kind of segment."

If diagrams were just ordinary segments, then a **CREF** frame configuration with a diagram in one half and text in the other would be like "two window mode" in **EMACS** (where two buffers can be selected at the same time, each using a different part of the screen). Unfortunately, a general facility for multiple windows is not easy to set up in the current **CREF** implementation because the underlying **ZWEI** data structures store segment connectivity information in the "node" (segment) rather than making it a property of the display area, forcing segments to always have a unique next and previous. To fix this problem would require either a substantial change to **ZWEI** or a decision to separate **CREF** from **ZWEI** altogether.

Diagram Types

CREF could use a great deal of work on its facilities for constructing graphic images such as diagrams. Experience with even the limited diagram facilities now available suggest that this is a rich and primarily untapped resource for augmenting the representation capabilities of the system.

For example, the only defined diagram types in the current implementation of **CREF** are `CHAIN`, `HIERARCHY`, and `AND/OR-TREE`. This set should obviously be extended.

Extensibility of Link Types

The set of link types used by **CREF** is easily extensible internally, but no user interface has been created for adding new link types (and commands to manipulate them).

With such extensibility will come issues of whether extended link types must be entered in some central registry so that different users do not use the same link name to mean different different things, or whether some notion of locality can adequately address that issue.

Diagram "Implications"

When filtering segments based on what keywords they have, a message passing protocol is used to query the selected diagram about whether a given set of explicit keywords for a segment imply the desired set of keywords. Each diagram type may provide its own strategy for how to answer this query; "hierarchies" may use simple subsumption, while "and/or trees" may use a more elaborate strategy. An illustrated example of this technique (using hierarchies) was given in Section IV.

In fact, however, pictures are only one way that such implications might be defined. The user might prefer to write logic rules or code. The system should later be extended to allow greater flexibility in how these implications are expressed.

Domains and Theories

There is some question about whether a domain and a theory are just instantiations of the same data structure at different levels. If that were so, then one might want to just make a general "context tree."

The current implementation uses a form of context tree but there are differences in how the levels are treated. We have described "domains" and "theories" as different kinds of things (rather than as the same kind of thing instantiated at different levels) to match the implementation.

For example, in CREF as it stands now, one may never select a domain without also selecting a theory in that domain. With careful thought (and some reimplementation), this restriction might usefully be removed.

The Order of Segments in an Abstract Collection

When the user names an abstract collection to be selected, a static collection is composed with the appropriate segments. The ordering of that collection is initially fairly arbitrary (related to the order in which the segments were created). It might be reasonable to use the PRECEDES relation to initially order such collections during such a selection (or to introduce some new ordering relation for presentation purposes).

Mutability of Collections vs Collection Names

Some operations in CREF create new collections, while others modify existing collections. For example, the command for tracing a reference link is Hyper-R; when it is typed, CREF selects a new collection containing the current node and its references. On the other hand, the command to trace a summary link, Hyper-S, will modify the current collection, replacing the current segment (which must be a summary) with the nodes it summarizes.

An unfortunate effect of this is that if a collection named "Overview" contains a number of segments which are summaries and those summaries are then expanded, the collection will still be called "Overview" even though the name may no longer be appropriate. The alternative would be to create a new collection called "Overview (Expanded)" or some such. But the problem is that expansion and contraction of summaries is a common operation and this would mean that the number of named collections would increase drastically.

In this sort of system, there is a never-ending tension between trying to name everything (in which case, the number of named things may grow quickly and the set may become quickly unmanageable) or to name as little as possible (in which case, things that took a lot of trouble to construct may be hard to retrieve if one accidentally drops the pointers to them).

Locality Issues

Editing features such as font maps and editor modes which are commonly associated with a buffer or file in EMACS or ZMACS do not have an obvious "home" in the non-linear model of text connectivity employed by CREF.

For example, they might be associated with collections. In that way, a certain segment might be edited in text mode in one context and lisp mode in another context. It would be more probable to suppose that such information should be maintained at the segment level. This could mean a potentially large amount of mode information would need to be carried with each segment.

A hybrid solution might say that there should be some kind of object which could serve as a mode descriptor for sets of segments (in some structure orthogonal to collection structure). A user might designate a default descriptor for new segments and could later modify either the default or the descriptor pointer for a node.

Only more experiments will say which solution is correct, but this problem is representative of the sort of issues that begin to crop up in a system which diverges from the traditional, linear, "single view" model of text.

Lack of Constraint in Connectivity

Another problem that relates to the expansion of summaries is illustrated by the following scenario: A user summarizes segments A, B, and C with a summary segment S1. Then he selects a collection by some match on the keyword, K. Suppose the selected collection happens to contain A and B but not C. Then if the user notices that A has a summary, he may run the command to replace A (and, incidentally, B) with S1. This is fine, but if he later expands S1, he will find that the buffer contains A, B, and C.

There is more than one approach to solving this. We might allow the summary contraction facility to only show summaries for nodes which have all their summarized nodes present in the current collection. Alternatively, we might provide daemons which do consistency checking on known constraints (such as that all things in the collection we just described should have the keyword K) and took some appropriate action to achieve the constraint or warn of its violation.

Bidirectionality of Link Types

CREF actually internally maintains link types REFERENCED-BY, SUMMARIZED-BY, SUPERSEDED-BY and PRECEDED-BY in order to provide backpointer information for its four primary link types (*i.e.*, REFERENCES, SUMMARIZES, SUPERSEDES, and PRECEDES).

The user may usefully think of links as simply bidirectional, although in fact forward and backward links are not always maintained in exactly one-to-one correspondence when superseding occurs. For example, if segment A summarizes segment B and segment B becomes superseded by B′, the resulting links will be:

```
A summarizes B′
B summarized-by A
B′ summarized-by A
B′ supersedes B
B superseded-by B′
```

Garbage Collection

In the current implementation of CREF, segments can be modified destructively. In order to save "back versions" of a segment, a command can be used to "freeze" a segment. The frozen segment is copied and a SUPERSEDES link is created to point from the copy to the original segment, which is then marked inactive.

Since pointers are retained from active structures to inactive ones, the process of garbage collection is complicated. The normal Lisp garbage collection mechanisms will not in general understand when inactive segments can be reclaimed. In some cases, this is desirable because the saved versions of segments may provide important historical information. Because of segmentation, the cost of retaining back versions is greatly reduced, but it still may be undesirable or impractical to retain all such back versions. Strategies for automatic garbage collection or tools for editing the pool of accessible objects is an area which deserves attention during the course of later work.

Knowledge Representation

Clearly there is much room for expansion of CREF to improve its usefulness as part of a KEA.

The bulk of CREF's facilities have thus far been directed toward the task of perusing bulk textual information with the aim of turning it into structured information. In the future, much more work needs to be directed at the problem of creating and manipulating structured knowledge bases such as those used by these other knowledge engineering systems. For example, the BT researchers have expressed a desire for facilities that notice "gaps" in their models of an expert behavior (where perhaps the expert has not been queried about a certain situation or component). This sort of capability is not currently supported by CREF, but is clearly essential to the broader goals of a KEA.

One interesting area is that of **semantic networks**. Much work has been done in relating arbitrary nodes in a graph using semantic networks. The network-like structure of CREF calls for interconnections of a character similar to those used in semantic networks. Experimentation needs to be done to assure that the right kinds of relational structures have been chosen for describing the interconnections of CREF segments.

Cooperating Workstations for Distributed Projects

Another unexplored area, but one which certainly presents itself as ripe for investigation, is the question of how workers on two different workstations (either timesharing or with hardware-level concurrency) could usefully work together on the same workspace at the same time. There are many aspects of **CREF**'s structure which bear strong resemblance to things found in conventional **database theory**, so it is reasonable to assume that database theory has something to teach **CREF** about process interlocking, *etc.* However, the key question for investigation would be what shape the interface would need to take so that users would not become bogged down in unnecessary protocol.

VIII. Summary

We have described **CREF**, a "cross referenced" editing facility which runs on the Lisp Machine.

CREF deals with chunks of text called **segments** which may have associated features such as **keywords** or various kinds of **links** to other **segments**.

Text in **CREF** is organized into linear **collections** for normal browsing. The use of summary and cross-reference links in **CREF** allows the imposition of an auxiliary **network structure** upon the text which can be useful for "zooming in and out" or "non-local transitions" (*e.g.*, to footnotes).

CREF also takes good advantage of modern graphical presentation and input technology by allowing users to create **diagrams** which express various kinds of semantic relationships between objects in a given application domain and then to use such diagrams to influence aspects of **CREF**'s behavior.

Although originally designed as a tool for **protocol analysis** in the **KEA**, the **CREF** editor is a general purpose facility which would be suitable for a number of other text editing tasks including **editing code** and **document preparation**, as well as some tasks not traditionally thought to be primitively supported by text editors, such as **mail reading** and **documentation browsing**.

Although **CREF** offers many capabilities not available in earlier text editors such as **EMACS** and **ZMACS**, it has a number of unusual problems that are artifacts of those extended capabilities. Those problems include issues of **presentation**, **extensibility**, **constraint**, and **garbage collection**.

There have been a few other text editors like **CREF** which manipulate "structured text," most notably **NLS**, **TEXTNET**, and **BOXER**. In spite of their interesting functionality, none of these projects has caught on and become the "editor of choice" for any significant community.

The point of this paper has not been to convince the reader that **CREF** should be the "editor of choice"; it has been to inspire people to at least consider how facilities such as those provided by **CREF** could change the character of their routine (and not-so-routine) editing — hopefully for the better. Perhaps from such thought will come change.

References

[Akscyn 82] R.M. **Akscyn** and D.L. **McCracken**, "The ZOG Project," CMU Computer Science Department, Pittsburgh, PA, draft June 1982.

[Bobrow 80] D.G. **Bobrow** and I.P. **Goldstein**, "Representing Design Alternatives," *Proceedings of the AISB-80 Conference on Artificial Intelligence*, Amsterdam, July 1980.

[Brachman 80] R.J. **Brachman**, "An Introduction to KL-ONE," Bolt Baranek and Newman, Inc., 50 Moulton St, Cambridge, MA 02138, draft April 1980.

[Ciccarelli] E.C. **Ciccarelli** and E.A. **Killian**, The **BABYL** program has no published reference, but exists on most **TOPS-20** systems which run **EMACS** and is documented under the **INFO** documentation system on those systems.

[diSessa 84] A. **diSessa**, "A Principled Design for an Integrated Computational Environment," MIT Laboratory for Computer Science, Cambridge, MA 02139, May 1984.

[Engelbart 68] D.C. **Engelbart** and W.K. **English**, "A Research Center for Augmenting Human Intellect," AFIPS Conference Proceedings, Vol. 33, Fall Joint Computer Conference, San Francisco, CA, December 1968.

[Engelbart 70] D.C. **Engelbart**, et al, *Advanced Intellect-Augmentation Techniques*, Final Report on SRI Project 7079, Stanford Research Institute, Menlo Park, CA 94025, July 1970.

[Genesereth 81a] M.R. **Genesereth**, R. **Greiner** and D.E. **Smith**, "MRS Manual," Memo HPP-81-6, Stanford Heuristic Programming Project, Stanford, CA 94305, December 1981.

[Genesereth 81b] M.R. **Genesereth**, "The Use of Hierarchical Design Models in the Automated Diagnosis of Computer Systems," Memo HPP-81-20, Stanford Heuristic Programming Project, Stanford, CA 94305, December 1981.

[Goldberg 83] A. **Goldberg** and D. **Robson**, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley Publishing Company, Reading, MA, 1983.

[Gray 77] J. **Gray**, *Notes on Data Base Operating Systems*, IBM Research Laboratory, San Jose, CA, Summer 1977.

[Handel 83] J. **Handel** and T.S. **Whitaker**, *Zmail Concepts and Techniques*, Document #990096, Symbolics, Inc., Cambridge, MA 02139, 1983.

[KEA 84] **Knowledge Based Systems Group (British Telecom)** and **Human Cognition Research Laboratory (The Open University)**, "A Knowledge Engineer's Assistant for Constructing Knowledge Based Fault Diagnosis Systems," A Proposal to the Alvey Directorate for a Collaborative IKBS Research Project, February 17, 1984.

[Kehler 84] T.P. **Kehler** and G.D. **Clemeson**, "An Application Development System for Expert Systems," *Systems and Software 34*:212-224, 1984.

[Knuth 83] D.A. **Knuth**, "The WEB System of Structured Documentation" (version 2), Stanford University, Stanford, CA 94305, July 1983.

[Kowalski 83] R.A. **Kowalski**, "Logic Programming," Proceedings of IFIP-83, 1983.

[Lowe 84] D. **Lowe**, "The Representation of Debate as a Basis for Information Storage and Retrieval," Stanford Computer Science Department, Stanford, CA 94305, January 1984.

[McMahon] M. **McMahon** and M. **Crispin**, *MM User's Manual*. This manual is neither published nor dated, but generally exists in a file called `DOC:MM.DOC` on any DEC **TOPS-20** system that runs the **MM** program.

[Minsky 74] M. **Minsky**, "A Framework for Representing Knowledge," Memo 306, MIT Artificial Intelligence Laboratory, Cambridge, MA 02139, June 1974.

[Nelson 81] T. **Nelson**, *Literary Machines*, 1981. This unusual book (about "hypertext") does not contain much information about how or where it was published, but does say that you can order copies from Ted Nelson, Box 128, Swarthmore, PA 19081.

[Newell 72] A. Newell and H.A. Simon, *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[Newell 82] A. Newell, "An On-Going Case Study in Technological Innovation," CMU Computer Science Department, Pittsburgh, PA, May 1982.

[Pitman 83] K. Pitman, "Interfacing to the Programmer's Apprentice," Working Paper 244, MIT Artificial Intelligence Laboratory, Cambridge, MA 02139, February 1983.

[Schoen 83] E. Schoen and R.G. Smith, "IMPULSE: A Display Oriented Editor for STROBE," *Proceedings of the National Conference on Artificial Intelligence* (AAAI-83), Washington, D.C., August 22-26, 1983.

[Stallman] R. Stallman. The **INFO** program has no published reference, but exists on most **TOPS-20** systems which run **EMACS** and is extensively self-documenting (*e.g.*, offers an online tutorial). When present, it can be invoked either via the `XINFO` command to the exec, or the `C-X I` command from within **EMACS**.

[Stallman 81] R. Stallman, "EMACS: The Extensible, Customizable, Self-Documenting Display Editor," Memo 519a, MIT Artificial Intelligence Laboratory, Cambridge, MA 02139, March 1981.

[Sterpe 84] P. Sterpe, "TEMPEST – A Template Editor for Structured Text," Working Paper 257, MIT Artificial Intelligence Laboratory, Cambridge, MA 02139, May 1984.

[Teitelman 78] W. Teitelman, *Interlisp Reference Manual*, Xerox Palo Alto Research Center, October 1978.

[Trigg 84] R.H. Trigg and M. Weiser, "TEXTNET: A Network-Based Approach to Text Handling," Department of Computer Science, University of Maryland, College Park, MD 20742, 1984.

[Waterman 73] D.A. Waterman and A. Newell, "PAS-II: An Interactive Task-Free Version of an Automatic Protocol Analysis System," *Advance Papers of the Third International Joint Conference on Artificial Intelligence* (IJCAI-73), Stanford, CA 94305, August 20-23, 1973.

[Weinreb 81] D. Weinreb and D. Moon, *Lisp Machine Manual*, 4th Edition, MIT Artificial Intelligence Laboratory, Cambridge, MA 02139, July 1981.

Acknowledgments

The **CREF** system was designed and implemented by the author at the Open University during June-July 1984. Support for that development was provided by the Alvey Directorate and the Science and Engineering Research Council (SERC). This work would not have been possible without the foundation that had been laid by those at the Open University and at British Telecom, who created the concept of the **KEA**. Marc Eisenstadt, of the Open University, is to be thanked especially for talking me into the project and then supervising it. The experience of Chas Church, Nikki Dick-Clelland and George Pollard at BT provided essential guidance and feedback which allowed me to understand their needs as knowledge engineers.

Dan Brotsky, Marc Eisenstadt, John Mallery, Fanya Montalvo, Chuck Rich, Peter Sterpe and Dick Waters read drafts of this paper and provided much useful feedback.

END

FILMED

10-85

DTIC