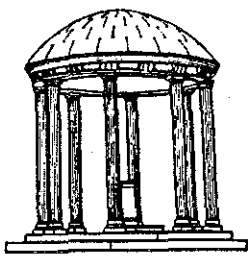
MICROARRAS: An Overview

TR86-017

August, 1986

John B. Smith, Stephen F. Weiss, Gordon J. Ferguson



The University of North Carolina at Chapel Hill
Department of Computer Science
Sitterson Hall, 083A
Chapel Hill, NC 27599-3175

A TextLab Report

Introduction

Text editors are now the standard means for producing all sorts of documents, from the shortest memo to complete books. Sophisticated word processing software facilitates the entire writing process, from creation and revision to production of the final copy. In fact, electronic text is becoming a new medium for communicating documents. The growth in electronic publishing combined with the maturing technology for converting printed text to electronic form (e.g., the Kurzweil Scanner) assure a large and rapidly growing pool of machine readable documents. When developments in optical disks are added to the equation, we can safely predict that individuals will soon be able to acquire, store, and access very large databases of textural information at a reasonable cost. What is now needed are more sophisticated systems that can operate on a variety of hardware, ranging from microcomputers to mainframes, to provide efficient effective management, retrieval, and analysis of textual data.

This paper describes one such system, called MICROARRAS, that we are developing at the University of North Carolina with funding from the National Endowment for the Humanities and additional funding from Northern Telecom. We have designed the system to operate on large, hard disk microcomputers, such as the IBM PC/AT, on professional workstations, such as the SUN, and on mainframes, such as the Vax. The system supports flexible, efficient retrieval of text from arbitrarily large textual databases in response to a wide variety of query types. It can perform various statistical and other analytic functions on text and is designed to permit easy addition of new analytic capabilities. The system also provides for distributed textual databases and a variety of user interfaces.

The goal of our development effort is to produce a "next generation" system that extends the number and kind of resources provided the knowledge worker whose working materials include documents. Consequently, we have based our system on two driving problems. The first is the sophisticated, often esoteric needs of academic professionals, ranging from literary scholars and anthropologists to chemists and computer scientists. Analytic tools that can determine subtle relationships within and among texts, such as

thematic or stylistic differences, can serve as the basis for a much broader collection of measures and functions for general use. Our second goal, then, is to develop an expanded (prototype) system for these more general applications, particularly office automation and document retrieval in research and development environments.

Existing Systems

Virtually all word processing systems provide some degree of retrieval function for the text being created. However, queries are typically very restricted (e.g. exact matching of a character string) and search time in large texts may be prohibitively long. Furthermore, most provide few, if any, analytic tools.

Systems designed specifically for retrieval and analysis of large full-text databases are a relatively new development, but are already having a significant influence in a variety of fields. For example, the LEXIS [1] and WESTLAW [2] systems have had a fundamental effect on legal research. The BRS-American Chemical Society system [3] may soon do the same for chemistry. The University of Chicago is making available a collection of some 1,500 full-length texts for French studies [4]. And the New York Times [5] offers its database of current journalism in a full-text form.

These systems are for the most part derived from a single early system, STAIRS [6]. The exception is the University of Chicago project which is using an earlier version of our system, called ARRAS [7]. While the STAIRS-based systems have made significant contributions, certain fundamental features limit what they can do now and what they will be able to do in the future. First, they offer no unifying concept of text and database; all rely on *ad hoc* combinations of file designs, access, and maintenance techniques. Second, they logically separate the multi-text database from the individual texts contained within it. Consequently, once a document or set of documents has been selected through a database search, the user cannot do secondary searches for other combinations of words in the selected documents. Finally, current production systems cannot efficiently provide the underlying access support needed to develop more powerful analytic functions. They do

not have the necessary "hooks" onto which new features can be added easily. We believe that the utility of full-text database systems would be increased greatly if the user could apply automatic indexing, content analysis, statistical, or AI-based models to the database and to the texts contained within it. MICROARRAS is our attempt to alleviate these and other restrictions.

In the remainder of this paper, we discuss MICROARRAS. Section 2 shows MICROARRAS from the user's perspective. This includes the search, retrieval, and analysis functions that are currently available. Section 3 presents the underlying architecture of the system including file organization, search strategies, and the formal language used for communication between the user interface and the underlying retrieval engine. We also show the provisions in MICROARRAS that allow easy expansion. Section 4 describes several future directions for MICROARRAS that we intend to explore.

User's View

Overview

In this section we describe the user's logical view of the system. By "logical" view we mean the mental model the user constructs in order to understand how the system operates. It does not mean the literal visual appearance of the user interface. (MICROARRAS is designed to support multiple user interfaces that can be tailored for different applications and different groups of users; this feature is described in more detail in the **Architectural View** section). The overall *environment* is described first followed by descriptions of six key concepts: *passages*, *text display*, *lexical display*, *categories*, *searching*, and *arithmetic analysis*.

Environment

We envision the user working on a microcomputer or professional workstation in a distributed computing environment in which some form of network links the user to other

users and to remote mainframes or other compute- and file-servers. MICROARRAS can be operated within a single, isolated microcomputer, but for the remainder of this description we will assume a distributed environment. In that environment, the user will have access to one or more file systems (e.g., a local hard disk, a remote file-server, and/or a remote mainframe file system) and to conventional text editors. He or she will also have access to one or more textual databases maintained by MICROARRAS. With MICROARRAS functions, the user can insert a text into one of the databases, delete a text, and move or copy a text from one database to another. The user can also logon to a remote system. Since MICROARRAS works on both mainframes and workstations/microcomputers, the user can work with texts stored in a remote database with MICROARRAS operating on that machine and the user's workstation functioning as an intelligent terminal; or the user can work with texts stored in a database on the user's own machine and operate MICROARRAS completely within that environment. Or he or she can transfer texts back and forth from one environment to the other. In the current version of the system, the user must direct these operations; but in future versions, the system will hide much of this from the user.

Working with a textual database is frequently an iterative process. While the user sometimes knows exactly which text he or she wishes to access, more often identifying relevant texts involves some form of search. MICROARRAS will provide two database search modes: bibliographic search and content search. The first uses conventional bibliographic methods to help the user locate documents by author, title, descriptive keywords, etc.; however, in addition to retrieving bibliographic citations, the search also builds internal pointers to the associated documents stored in the textual database. In some cases, the search may locate a single document, but frequently it produces a list of candidate documents (e.g., the documents with certain words in their titles). In content search, MICROARRAS will support full Boolean search for words or sets of words to produce a similar candidate list.

Once a candidate list of documents has been identified, MICROARRAS provides extensive capabilities for secondary searches of selected documents (or parts of documents)

from this list. In the remainder of this paper we will emphasize MICROARRAS's secondary search and analysis capabilities since they represent some of the more unusual aspects of the system. Thus, we will presume that the user has identified a candidate list of texts and now wishes to begin narrowing the focus to find or analyze the specific passages relevant to his or her needs. A key concept in this process is the notion of a textual passage.

Passages

Passages are named sets of texts or portions of texts. They are used to direct MICROARRAS's attention for subsequent searches and analyses. For example, following a global search of the textual database, the user interested in structure editors might select five technical reports, two journal articles, a book, and two papers contained in the proceedings of a conference and then name the group ***structure-editors***. The reports, articles, and book are likely to be complete documents in the database, but the two conference papers would be contained within a proceedings document. The user could now retrieve and analyze text in this ***structure-editor*** passage. At any time the user may shift the focus of the analysis to another set of texts (or portions of texts). Thus, during a session, the user may create (and save for subsequent use) many different passages and easily move from one to another.

One of MICROARRAS's strengths is its flexibility in handling context specifications. The system generally views a document as composed of two overlapping hierarchies of text segments. The first denotes logical divisions, such as volume, chapter, paragraph, sentence, and word within sentence; the second, physical divisions such as page, line, and word within line. Whenever a textual database is established, the definition includes sets of canonical segments for that collection. ***Ad hoc*** segmentations may be used for individual documents, but the user must define any hierarchical relations that are to be recognized. These segmentation schemes are used by MICROARRAS to format the display. They are also used in a number of other operations, such as searching, that involve context. Some of these are discussed below.

Text Display

Having established some set of passages (identifying one or more documents or parts of documents), the user can quickly and easily move around in them. This is done by first specifying the desired passage and then indicating the place in that passage to be displayed. At a system level, indicating a place for display is done by specifying the appropriate segment, as described above. At the user interface level, the user might indicate that he or she would like to see the first section, entitled *Overview*, in the second journal article within the passage. We are currently designing a graphics-based interface in which this information may be selected directly from a tree diagram showing the segmentation hierarchy for a given document or passage.

A second way of moving around in the text is by displaying all occurrences of a given word along with some designated context around each. Essentially, this is an interactive concordance or keyword in context (kwic) listing. MICROARRAS can produce such displays immediately, no matter how large the document or how scattered the occurrences. The context around each occurrence can be varied at will, from one or two words on each side to the full section or even the entire document. If the user wishes to see a wider context for any particular occurrence, he or she may simply point to that occurrence on the display and ask for additional context. Thus, a typical user might specify a rather limited context – say, four or five words on each side – and then ask for additional context for those passages that seem most relevant.

Lexical Display

Just as MICROARRAS can immediately go to any place in a document to display text, it can also go to any place in an alphabetical lexicon of the words that appear in a document or passage. Lexical information may be displayed in three primary ways: by alphabetic sequence, by pattern of characters, and by frequency of occurrence in the document or passage. In the first instance, the user may display the lexicon or part of it by indicating an alphabetical range, such as all those words beginning with the letters

a through *c*. The display can be shown separately for each document within the current passage or interleaved to form a single alphabetic sequence. The second option searches for sequences of characters, including "wildcards". Thus, the user can locate all words with a given stem or a given prefix or suffix. The third option displays word-types sorted by their frequencies within the passage. The user can then locate the most frequent words, the least frequent, or those falling within some portion of the frequency spectrum.

Categories

In text analysis, it is often useful to define a group of words and then refer to that group by a single name. We call such groups *categories*. MICROARRAS supports three types of categories: type lists, token lists, and recursive lists. Type lists represent all occurrences of a specified set of word types for a particular passage; for example, each instance of the word-types *processor* and *cpu*. Token lists are sets of text positions representing individual occurrences of words; for example, the specific places in a document where *processor* refers to a computer, not a human processor of information. Recursive categories are sets of other categories; for example, a new category consisting of the *cpu-category* and the *memory-category*.

A category definition includes the name by which it is identified and the expression that defines it. The expression can be a list of word-types, tokens, or category names. (It can also be a Boolean expression, as described in the next section.) Actual internal definition of a category is a two-step process: the expression is scanned and stored internally; then it is evaluated with respect to a given passage. Thus, for example, a set of categories may be derived from a thesaurus that has been tailored to the user's research interests. When the categories in the thesaurus are initially defined (or obtained) by the user, they are stored as expressions (sets of word-types) independent of any document or set of documents. But when they are applied to a given text (or passage), the expressions are evaluated with respect to that document. Consequently, word-types that are in the category but don't appear in the text (or passage) are eliminated from the internal working instance of that category.

Search

MICROARRAS provides very flexible and very fast search of a document. To conduct a search, the user must specify three components: a Boolean contextual expression, a passage in which to evaluate the expression, and a category in which to store the resulting set of positions where the expression occurs (a token list). The expression is any Boolean combination of words or category names. If categories are used, they imply every occurrence of any word-type included. Contexts in which search expressions are evaluated can be specified in terms of any text segment units valid for the text database and can be specified in any number. Thus, one can look for all occurrences of *cpu-category* & *memory-category* within, say, three words of one another, or within the same sentence, or within three sentences, or in the same sub-section, etc. Different contexts can also be specified for subexpressions (e.g., *cpu-category* & *memory category* within three words of one another but not in the same sentence with *hardware-category*). The result of a search is a list of text locations. They are stored as a token-list category, as described above.

Like any other category, token-lists resulting from a search can be used in any way that categories are normally used, including within a subsequent search expression. Thus, the user may search for patterns of words, patterns of patterns, patterns of patterns of patterns, etc. Search expressions can also be saved from one session to another and they can be applied to different passages. Consequently the user can develop expressions that define a specific set of interests and use them at will on different documents or sets of documents.

Arithmetic Functions

MICROARRAS provides facilities to compute various textual measures and then display the results, analyze then with an internal arithmetic interpreter, or pass then to an external statistical analysis package. Two basic kinds of data can be computed. The first is frequencies of words or sets of words (categories) within a specified passage. Thus, the user can compute and display the frequency with which a word or category appears in a

document, a set of documents, or some part of a document. For example, the user can display the number times the words in the *cpu-category* appear in the passage described above: e.g., in five technical reports, the two articles, the book, and the two conference papers. The second major class of statistical data are segmental measures. These determine the number of times one segmental measure occurs within another. For example, the number of words in a sentence or the sentences in a document.

These measures become more interesting when combined in various ways to compute ratios, distributions, and lists of various kinds such as the distribution of a word or category over a text passage divided into a set of uniform intervals. The resulting vector of values can be displayed as a bar graph or passed out for statistical analysis. For example, one can perform a Fourier analysis on the vector to see if the word or category tends to appear at regular intervals. Or one can accumulate several such vectors, view them as the columns of a matrix, and perform a factor analysis to identify clusters of words or categories that consistently appear together in the passage.

These measures can also be stored as numeric variables. Types recognized by MICROARRAS are scalers and vectors of both integers and reals. These variables can be used in arithmetic expressions to compute any statistic for which the user can write the equation. For example, one can evaluate in real time a number of different retrieval formulae to compare the matches between search requests and sets of documents identified.

Architectural View

Overview

In this section, we provide a general overview of the system's architecture and then describe, briefly, some of its more unusual features. The discussion is divided into two main parts: the preparation of a document for use by MICROARRAS and the structure of the system itself.

Document Flow

Preparing a document for use by MICROARRAS is a four step process (see Figure 1). First, it must be transcribed into a machine readable form with internal marks indicating segments (boundaries, such as chapters, sections, and paragraphs). Second, it is converted into a canonical form, identical to that reconstructed by MICROARRAS for textual display. Third, it is scanned and inverted. Finally, the inverted text is inserted into the textual database.

The machine readable form can come from several different sources. It can be text written with a conventional word processor or editor. It can be extracted off a network. It could be text "read" by an optical scanner, such as the Kurzweil Data Entry Machine. Since MICROARRAS indexes major segment boundaries, it expects to see marks in the text that indicate chapters, sections, paragraphs, etc. Currently, MICROARRAS recognizes marks that follow TEX conventions. While MICROARRAS does not support full TEX, a text that includes macros denoting these features can be recognized. We will extend the conventions accepted by MICROARRAS in the near future to include popular word processing software, such as Microsoft Word, NROFF, and Script.

The machine readable text is processed by a prescan program to produce a canonical form. This format is identical to that produced by the analytic engine during text reconstruction and sent to the interface for display. It provides a formal means for MICROARRAS to recognize similarities in segmental structures for documents encoded in different systems as well as true idiosyncrasies of documents. This version of the text is still readable but is highly structured and is well-defined in terms of format conventions. As such, it also serves as a portable form for transferring documents from one environment to another.

The canonical form of the document is then scanned. During this stage, the text is inverted and a number of separate indices created. The details of these indices are discussed in relation to the Analytic Engine, below. The result of this process is a single

"file" that constitutes the inverted text and all its associated indices. This form, again, is well defined and can be transported from one environment to another.

Finally, the inverted document is inserted into the textual database. This operation includes adding a citation to the bibliographic database for subsequent search. We are also in the process of developing facilities for consolidating the lexicons for the individual documents into a composite lexicon for the entire textual database. Once a document is in the database, it may be analyzed directly or it may be extracted in inverted form and transmitted to another MICROARRAS database or another system.

System View

Overview

¿From a development standpoint, our immediate goal was to convert a text analysis system running on large IBM mainframes (in PL/1) to a microcomputer version in C. We took advantage of the opportunity to substantially rethink and redesign the system. Not only have we extended its function, we have also broached fundamental questions about the formal models underlying text analysis and recast some traditional questions about efficient representation schemes and search algorithms for large textual databases. The next several sections describe the MICROARRAS implementation and touch on these larger questions.

MICROARRAS has three main modules linked – but also separated – by two well-defined interfaces (see Figure 2). Properly speaking MICROARRAS is a family of systems - each distinguished by a particular interface. The module that controls the User Interface interacts with a Command Processor module using a formal two-way language. This language provides a virtual machine, or facade, to the User Interface that we call FLANGE (Facade Language). The Command Processor parses FLANGE, checks it for errors, and then calls on the Analytic Engine to compute results. The Analytic Engine can be viewed as a collection of abstract data types; we treat the C functions that provide access to these sub-modules as a language, which we call Arrish. Thus, the typical cycle of operation is for the

User Interface to transmit FLANGE to the Command Processor. The Command Processor interprets the FLANGE command to produce a sequence of calls to the Analytic Engine. The Analytic Engine does the work and returns the data to the Command Processor which encodes it into FLANGE and sends it back to the User Interface. The User Interface unpacks the FLANGE-encoded data and displays the results.

The User Interface Module

Had we followed a conservative, top down approach to designing MICROARRAS, we would have first specified a User Interface and then incrementally elaborated the functions necessary to support that specification. Rather, the actual emphasis in our project was on designing a system to be viable across a range of different user environments. The relative portability of C code made this a reasonable goal and the volatility of the computer market made it almost a necessity. We wanted to run not only on microcomputers but on professional workstations, minicomputers, and mainframes, as well. MICROARRAS currently runs on the PC/AT (with the EGA Card) under MS-DOS, on the SUN workstation, and on the VAX 11/785 – the last two under BSD 4.2 UNIX. The fruits of this approach include the three quite distinct User Interfaces described below.

The first is a forms based User Interface on the PC/AT using the Lattice Windows package and the Panel software from Round Hill Computer Systems, Ltd. This interface directly supports the FLANGE concepts and functions outlined above. Its intended purposes are to drive the Command Processor and Analytic Engine for system testing and for helping the user learn FLANGE for developing other systems that can interact with textual databases.

A menu of commands is shown at the top of the screen. Command parameters are entered in a separate area of the screen (window). Results are displayed in another. Error messages are displayed in a fourth fixed area. A category, for, example is defined by filling in a single form; concordances can be requested using another. The user creates and sends

commands to MICROARRAS one at a time. Each command is processed and the results displayed immediately.

The great advantage of a forms-based user interface like this one over a command language is user convenience. The user can *select* an available option rather than having to first learn all the things the system can do. Thus, necessary control information can be entered on the associated forms. The on-line help is also keyed to particular fields in particular forms, errors can be pinpointed exactly, and commands altered by changing a single highlighted field.

The second interface was also done for the PC/AT but within Microsoft's Windows. Whereas the first interface was designed around the structure of FLANGE with an approximate one-to-one correspondence between user commands and FLANGE commands, the second interface is designed around the kinds of operations that users will typically perform. There is frequently a one-to-three or -four ratio between user command and those of FLANGE. Thus, we are beginning to use FLANGE as a programming language. This interface is also menu-based and was designed to include a minimum set of control features. One particular feature worth noting is its ability to transfer parameters from one window to another. For example, the user can display a concordance, select a particular occurrence, and then transfer the word token identifier to a second window in which he or she is constructing an analytic category of text positions. While this interface in considerably more powerful than the first and will be put into actual use, it will also serve as a stepping stone to a third.

We are currently designing a more sophisticated graphics-based User Interface. It will be highly iconic and support a visual command language. Features include:

- Iconic representation of textual objects – words, categories, contexts, Boolean configurations, etc.
- Convenient visual tools for combining and manipulating textual objects.

- A tree drawing of the text – showing the chapter and section organization – through which the user can navigate using the mouse to select portions for viewing or for further analysis.
- Separate windows for reviewing the current state of the system – the categories that are active, the format options available, etc. These can be opened or closed at will.
- Graphs (bar or line) of frequency distributions and other statistical data computed by the system.
- Formatted text showing bold face and special fonts for titles, etc; this text can be scrolled through and also stored for later display.

By using FLANGE as a primitive programming language, we can develop a succession of more sophisticated interfaces as well as specialized interfaces tailored to particular groups of users or particular applications.

FLANGE

FLANGE serves two major functions: it provides a formal specification for the MICROARRAS System and it provides an internal two-way command language. Formal specification of MICROARRAS was desirable for several reasons. MICROARRAS is part of an ongoing research project in natural language and text analysis. Consequently, we need to build tools that will outlive particular systems. We also envision MICROARRAS as one component of a larger system. MICROARRAS is intended to run on one or more nodes in a distributed text analysis network. We are also developing an expert system component to support intelligent user functions – e.g., more powerful search facilities. MICROARRAS will be used in that configuration as a compute-server communicating with the expert system through FLANGE.

FLANGE is based on the command language of mainframe ARRAS. But we have made several substantial extensions. FLANGE has a formal syntax; its grammar is specified using a BNF-like notation. Consequently, programs can easily construct command

expressions which, in turn, can easily be parsed. The components of a FLANGE sentence are strongly typed to simplify processing and to ensure reliable transmission across a communication interface.

FLANGE is used for all communications between a User Interface and the processing modules. Thus, for example, a request for a concordance is encoded into FLANGE by the User Interface. Then, as the results are computed they are encoded within the conventions of the "return" part of FLANGE and sent to the User Interface for display. Opportunities for interruption and cancellation of long outputs as well as error reporting are provided. Thus, FLANGE makes it possible to run the processing modules on one system and to drive a User Interface through a communications link running on another. We are developing logic to support multiple users sharing access to a single processing module and to view these users and support modules as nodes in a general network.

Command Processor

The second major component of MICROARRAS is the Command Processor module. It examines the FLANGE message for errors and then calls Arrish functions to actually perform the computations. In addition, the Command Processor contains the code to actually receive and transmit FLANGE to the User Interface. This section describes some of the techniques used in the Command Processor and identifies areas of continuing interest.

Although FLANGE is a formal language, no effort was made to generate a parser automatically. Each command has a small hand written parser; because the syntax is simple, these are easy to build without tables or external data structures. The advantage of using custom parsers on this constrained syntax is that detailed error diagnostics can be generated. Typically an error can be located by command line and syllable and can be identified as the wrong type of syllable or as a syllable whose contents are incorrect. The balance between detailed error handling and the elegance of language constructs continues to be a major issue as we consider new versions of FLANGE.

During execution the command is viewed as an expression to be evaluated. The evaluation entails building a parameter list and then calling functions provided by the Analytic Engine. Often this requires evaluating separate subexpressions and then combining their individual results. For example, the command to create a passage may designate several portions of a text – each by volume, chapter, and section. The command processor must first translate 'chapter', 'volume', and 'section' into their internal codes, then find the beginning and end of the first section, compute in a similar way the boundaries for the others, then combine these to form the passage.

Some evaluations may require several steps. A concordance, say, might be requested for a list of words. The command processor would first locate all the occurrences of the word, then recreate the context requested, and finally return the textual data to the User Interface via FLANGE. Intelligent ordering of these operations can improve system performance; such optimizations are of increasing concern for large collections of documents.

The Command Processor maintains a symbol table, providing a name space for FLANGE. Consequently, objects can not only be defined and reused during a session, they can be reconstructed through FLANGE commands stored in a file. This allows saving the complete state of a session on secondary storage. It also allows communication between two MICROARRAS programs.

In light of our interest in FLANGE as a formal description of text analysis, the routines used by the Command Processor take on considerable significance. These functions get the job done, but they also constitute a second formal description of text. The next section introduces that perspective.

Arrish

Arrish is the symbolic interface between the Command Processor and the Analytic Engine that does the actual computing and data manipulation. That interface can be viewed as a set of abstract data types, implemented in accord with good software engineering practices. However, it is more interesting to think of them as 'objects' implemented

in C. While this says that text analysis programs can be written in an object-oriented style, it also suggests that text analysis itself can be described in an object-oriented way. An exciting perspective of the Arrish language is that it molds the user's ideas about the formal properties of texts. That is, it provides a set of textual primitives that can be used not just in computing measures of a text but for *thinking* about what constitutes a text. Thus, it is a step toward defining a *text* processing language, rather than a *string* processing language. For example, the user can think formally about the distribution of certain classes of words across various sections of a document, the shifting of sentence and paragraph lengths as an author matures, or about various patterns or rhythms of concept co-occurrence across a text.

Analytic Engine

The Analytic Engine (sometimes called the Arrish Engine) is embodied in some dozen abstract data types: *spans*, *passages*, *mark-sets*, *segments-in-effect tables*, *type lists*, *token lists*, *text access* and the like (Figure 3 shows the basic system modules and their dependencies). These modules contain text and data, with no (known) hidden iterations. This provides security and reliability at the cost of interfering occasionally with some kinds of optimizations. However, we believe these problems can be overcome in future refinements of the systems.

The actual text is stored as an inverted file. Since the original text format is not kept, any text displayed is reconstructed from this index structure. The index structure consists of three main components. The *dictionary* is an alphabetic list of every word-type occurring in the text, its frequency, and a pointer to an occurrence list. The *occurrence list* contains the position of each word token in the sequence of words that constitute the text. A *linear image* of the text contains an entry for each token position in the original text. The integer in the i-th position of the linear image is the index into the dictionary of the word in that position. In addition to these main components, there are several secondary structures. Each position in the text has an associated format code for

upper/lower case information. Additional structures index the segmental organization of the document (e.g., chapters, paragraphs, sentences). For each segment mark there is a list of all its occurrences. Segment titles (e.g., chapter titles) are stored in a string table. These structures for a single text are stored contiguously within the database file. That file also includes header blocks which locate these various components for a particular text. MICROARRAS reads this header data at the start of a session but accesses textual data as it is needed. The buffering techniques are beyond the scope of this description.

The abstract data types provide both a symbolic interface to these data as well as the actual computational functions to access or to use them. In the current version of the system, data structures are determined at the time a document is inserted into the database. Consequently, examining multiple documents for secondary searches is done iteratively. We are developing algorithms for extracting and consolidating collections of documents into integrated data structures. This will permit the same immediate access to larger collections that is now possible for small collections.

Future Plans

As we have mentioned throughout, MICROARRAS is part of a continuing program of research in natural language and text analysis. Below we describe four areas of future development directly linked to MICROARRAS.

We are particularly interested in techniques for building large textual databases using emerging low-cost, high-volume media. Especially attractive are the new write-once optical disks. We are exploring alternative strategies for indexing large volumes of textual data appropriate for these devices. We are also interested in efficient means for transferring documents from one environment to another, strategies for determining relative efficiencies for local vs. remote processing, and other issues that accrue from recent advances in storage and networking technologies.

In describing the MICROARRAS engine, we mentioned that we use the two-way communication language, FLANGE, as a primitive programming language. That is, the User

Interface interprets the user's intentions and then constructs commands in FLANGE which are sent to the Analytic Engine for execution. Results are packaged in FLANGE and send back to the Interface for display. We plan to reconsider FLANGE with the intention of turning it into a true object-oriented programming language for text processing and analysis. While several string-processing languages (e.g., SNOBOL) have been viewed as especially appropriate for text applications, they are *string* languages, not *text* languages. A true text language would provide as primitive objects such entities as words, sets of words, sentences, paragraphs, documents, sets of documents, etc. Primitive operations would include search, pattern-matching, counting, associating, and other functions. Implemented as an interpreter, the language would become a high-level general purpose programming language for text applications analogous to APL for numeric applications. Implemented as a compiler, it could be used as a development language for building specialized text processing systems.

We are currently developing a graphics-based structure editor for helping authors during the design and drafting stages of writing. The system is an extension of the outline processors such as Thinktank. Instead of working within the linguistic structure of the outline, users of our system work within an abstract space of nodes and arcs, where nodes represent concepts and arcs associations between them. The writer transforms a network of concepts into a hierarchy and then writes the document by writing blocks of text for each node in the tree. The result is better management of the writing process and documents with more coherent structures.

After we complete basic versions of MICROARRAS and the structure editor, we will merge the two systems to form a comprehensive environment for the textual knowledge worker. That is, the user – particularly during the exploratory and planning stages of writing – will be able to search the MICROARRAS textual database. When relevant passages are located, he or she will import them into the writing environment, encapsulate them within a node, and link that node into the overall structure of ideas being built. When the document being written is complete, the user will transfer it from the structure editor to the MICROARRAS textual database where it will become another document

that can be searched and used, by the original user but also by other workers with access to the database. Thus, we will complete the cycle of document creation, storage, search, retrieval, and reuse.

Finally, we have begun developing intelligent functions to be embedded in the integrated environment just described. As we have stressed throughout, the FLANGE two-way communication language gives us considerable flexibility. Just as different interface programs can present different visual appearances to the user but still translate the user's expressed intentions into FLANGE commands, so we can write other programs that can communicate with the MICROARRAS engine through FLANGE. One such program we are developing is an intelligent search function using an expert system as a compute server. The problem we are addressing is the laboriousness of searching a full text database in which the user must compose long, often complex Boolean contextual search expressions, examine the resulting documents, decide what is important and what is not, store relevant passages for future use, and then repeat the process to follow new ideas as they emerge. We believe that at the exploratory stage of thinking, a network of associated concepts may model the user's intentions more accurately than more precise Boolean expressions. In the system we are developing, the user will first sketch an associative network of concepts using the structure editor. The expert system will then expand those concepts into categories of words and phrases using a thesaurus tailored to that individual's interests. It will then form search expressions in FLANGE, submit them to MICROARRAS, and analyze the resulting passages. The system will then rank them according to relevance criteria, update the associative network to indicate additional concepts found in the retrieved passages but not in the original network, and otherwise mediate the user's encounter with the system. This work is progressing and we expect to have concrete results during 1987.

In our continuing program of research, MICROARRAS constitutes a major cornerstone. We hope that others will find it a useful tool. But it will also serve as a foundation for future work.

Notes:

[1] *Lexis Handbook* (Interim Version), New York: Mead Data Central, Inc., 1980.

[2] *Westlaw Reference Manual: Revised Edition*, St. Paul: Westlaw Publishing Co., Inc., 1982.

[3] User's Guide: *American Chemical Society Experimental Full-text Primary Journal Database*, Columbus: American Chemical Society, 1981.

[4] Morrissey, R. and Del Vigna, C., *A Large Natural Language Database: American and French Research on the Treasury of the French Language*, EDUCOM Bulletin 18, 1 (Spring 1983).

[5] *The Information Bank-11: BRS/SEARCH Protocol User Guide*, New York: The New York Times, 1981.

[6] *STAIRS/VS: General Information*, IBM publication 12-GH 12-5114-2, New York: The IBM Corp., Inc., 1974.

[7] Smith, J.B., *ARRAS*, Perspectives in Computing 4, 2/3 (Summer/Fall, 1984).