The Text Analyzer: a Tool for Knowledge Acquisition from Texts

by

Judy Kavanagh

Thesis submitted to the School of Graduate Studies and Research

of the University of Ottawa

in partial fulfillment of the requirements for the

Master's degree in Computer Science

Under the auspices of the

Ottawa-Carleton Institute for Computer Science

Department of Computer Science

University of Ottawa

Ottawa, Ontario, Canada

August, 1995

ISBN 0-612-11565-8

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Acknowledgments

First I would like to thank my supervisor Dr. Doug Skuce for his knowledge, guidance and friendship during my time in the AI Lab. Thanks for the piano too!

Secondly, I would like to thank Ingrid Meyer of the School of Translators and Interpreters for her input from a linguist's point of view and for encouraging her students to use my Text Analyzer in their research.

Thanks to all of the other students in the AI lab, especially Noreen Harrigan for helping with the Icon programming, and Lynne Bowker and Laura Proctor for giving me valuable feedback on the Text Analyzer. Thanks also to Charles Tran, Kristen Mackintosh, Krista Kelly, Tim Lethbridge, Ken Iisaka, Michelle Ganley and the others for making the AI lab a fun place to work.

I would also like to thank the Natural Sciences and Engineering Research Council for awarding me a scholarship that has allowed me to concentrate on my research without financial concerns.

Finally I would like to thank my friends, human and otherwise – Don and Val, for giving me a life and for keeping me sane, and Rossi, Pie, Callie and Spotty for their love and companionship.

Abstract

The world is being inundated with knowledge at an ever-increasing rate. As intelligent beings and users of knowledge, we must find new ways to locate particular items of information in this huge reservoir of knowledge or we will soon be overwhelmed with enormous quantities of documents that no one any longer has time to read.

The vast majority of knowledge is still being stored in conventional text written in natural language, such as books and articles, rather than in more "advanced" forms like knowledge bases. With more and more of these texts being stored on-line rather than solely in print, an opportunity exists to make use of the power of the computer to aid in the location and analysis of knowledge in on-line texts.

We propose a tool to do this – the Text Analyzer. We have combined methods from computational linguistics and artificial intelligence to provide the users of the Text Analyzer with a variety of options for finding information in documents, verifying the consistency of this information, performing word and conceptual analyses and other operations. Parsing and indexing are not used in the Text Analyzer. The Text Analyzer can be connected to CODE4, a knowledge management system, so that a knowledge base can be constructed as knowledge is found in the text. We believe this tool will be especially useful for linguists, knowledge engineers, and document specialists.

Table of Contents

List of Figures

1 Introduction

In this chapter we introduce the problem of extracting knowledge from texts and introduce the Text Analyzer, a program designed to help solve this problem. We describe who might use the Text Analyzer and describe the goals and structure of the thesis.

1.1 Extracting knowledge from texts

Knowledge is arguably the prime resource of today's world and more and more knowledge is being generated and written down every day. Although enthusiastic AI researchers have tried to convince people to store knowledge in knowledge bases [Skuce and Lethbridge, 1995 to appear] [Lenat et al., 1990], this idea has not yet become popular. The vast majority of knowledge is still stored in natural language texts such as books, articles, and on-line documents.

A major problem with using natural language texts to store knowledge is that someone who wants to use that knowledge may have trouble finding it in the text. Natural language is wordy and many of the words do not contain actual information but are simply there to make the text grammatical. Reading the text, therefore, takes a lot of time. Another problem is that the subject of interest may be scattered through several different texts with a lot of uninteresting text in between. Again, it takes time to read through all the text to locate the sections of interest.

This thesis presents a step towards a possible solution to the problem of finding knowledge in a text. We have designed and implemented a computer application called the Text Analyzer to assist people in locating and analyzing the knowledge in documents quickly and efficiently. The Text Analyzer consists of a user interface on top of a set of operations that perform various tasks such as finding all the occurrences of a word in a text, ranking the words in a text by frequency, and suggesting possible superconcepts for the concept denoted by a word. The Text Analyzer may be used alone or it may be connected to the CODE4 Knowledge Management System so that a knowledge base can be constructed from the knowledge found in the text.

With the increasing number of documents that are now on-line, we believe that this type of tool could become as important as word processors are today.

1.2 Who could use the Text Analyzer?

It is our belief that the Text Analyzer could be used by many different types of people whose jobs require them to search for knowledge in documents but we think that it would be especially useful in the following fields:

1. Linguistics – *Terminologists* (who study the words, known as *terms*, that are used in a particular domain), *lexicographers* (who compile dictionaries), and *translators* all must examine large quantities of text, often in the millions of words, in order to find evidence of word usage [Burnard, 1992]. The Text Analyzer is designed to handle large texts and to perform many of the tasks that linguists do currently by reading these texts and writing the results on little pieces of paper.

2. Knowledge engineering – To build knowledge bases, expert systems and other knowledge-based systems, knowledge engineers must study documents and transcripts of interviews with experts. Knowledge from different sources must be compared for consistency [Keller, 1987]. These are procedures that could be aided by the Text Analyzer.

3. Document writing – Technical writers have the unenviable job of converting stacks of reports written, often unwillingly, by non-writers into readable and correct documents. They must cope with both inconsistent or ambiguous terminology and fuzzy explanations. The Text Analyzer could ease the pain of sorting out these texts. It could also help with the editing of existing documents in order to search for inconsistencies in both the document and in the underlying concepts.

1.3 Goals of the thesis

The first goal of this thesis was to design and build the Text Analyzer, consisting of a set of text processing tools, a user interface and a connection to the CODE4 Knowledge

Management System. The Text Analyzer was intended to bring together a variety of tools and analysis methods not previously combined in one tool, thus providing a system to compare the different approaches to document analysis.

The second goal was to test if a tool like the Text Analyzer would be useful in practice. We used the Text Analyzer to analyze a large text and build a knowledge base from the information found. An account of some of this process can be found in chapter 4. Two Ph.D. students in Linguistics are also using the Text Analyzer to aid their research.

We were also interested in seeing if the Text Analyzer could be constructed to work accurately and efficiently without parsing the text or indexing the documents.

1.4 Organization of the thesis

Chapter 2 of the thesis investigates related work in the fields of linguistics, artificial intelligence (knowledge acquisition in particular) and finally in the new and growing field where these two areas intersect. Existing systems that perform some of the operations of the Text Analyzer are discussed as the inspiration for our work.

Chapter 3 presents the Text Analyzer itself and discusses the interface, the operations and the general methodology of using it.

Chapter 4 discusses implementation details of the programs.

Chapter 5 contains a detailed example of using the Text Processor on an actual text about operating systems. It shows how a knowledge base was built containing the knowledge in the text.

Finally, Chapter 6 provides a summary of the thesis, discusses problems, and presents future work to be done on the Text Analyzer.

2 Related work

In this chapter we discuss research that the Text Analyzer draws on from the fields of linguistics and computer science. First we look at the area of *corpus linguistics* which examines techniques for analyzing large bodies of text for the purposes of linguistic analysis. Next we examine some computerized tools designed for linguists that use the principles of corpus linguistics. Then, from computer science, we examine knowledge management systems that organize and store knowledge. Finally we look at systems that attempt to link some of the techniques of corpus linguistics and knowledge management together.

2.1 Corpus linguistics

In linguistics, a *corpus* (plural *corpora*) may be defined as "a collection of texts assumed to be representative of a given language, dialect, or other subset of a language, to be used for linguistic analysis" [Francis, 1982]. The advantage of using a corpus for research is that it provides ready-made examples of word usage that can be put in a dictionary entry or cited in a study of a particular word.

If a corpus is large enough, it can also can be used as a basis for statistical studies of language. It could help answer questions such as "What is the most common word in English?", "Which word is used more often: *sofa* or *couch*?" Obviously, the larger the corpora, the more solid the evidence that answers such questions. Because examining a large corpus involves processing huge amounts of data, the ideal tool for handling it is the computer. Although corpora have been used in linguistics research for many years, a change took place in the early 1960's when the first computerized corpus appeared. This has led to an increasing amount of research being done in the field of corpus linguistics, especially in the last five years or so.

Currently, corpora may contain tens or even hundreds of millions or more words and they are getting bigger all the time as computer storage becomes less expensive. A corpus of several millions of words is large enough to contain enough examples of the word

or feature being studied to be statistically significant. [Aijmer and Altenberg, 1991] [Leech, 1991]

The first computer corpus of English to be assembled was the Brown Corpus which was compiled at Brown University during 1961 to 1964. It contains approximately one million words divided into 500 text samples of about 2,000 words each on 15 text type categories taken from material published in 1961 [Francis and Kucera, 1982]. The Brown Corpus is still widely used today. Many other computer corpora are available, containing written and spoken text in various languages.

Aijmer and Altenberg in [Aijmer and Altenberg, 1991] describe a number of well-known computerized corpora of English texts including the Birmingham Collection of English Text (more than 20 million words), the TOSCA Corpus (1.5 million words), the Lancaster-Oslo/Bergen Corpus (1 million words), the Longman/Lancaster English Language Corpus (30 million words), and others.

Early corpora were typed into a computer manually but modern technology allows text to be automatically scanned from printed material or assembled from text already in machine-readable form [Leech and Fligelstone, 1992].

Corpora are usually assembled carefully, containing texts from a variety of sources and about a variety of subjects in order to make them truly representative of the language. As well as words and punctuation marks, corpora may contain additional information such as annotations about the origin of the text samples, part of speech tags for each word, syntactic tags indicating sentence structure, or semantic tags marking word meanings.

The uses of a computerized corpus are many and varied. In linguistics a corpus provides evidence and examples of word usage. In natural language processing a corpus provides a body of unrestricted text on which to train and test language processing systems. Corpora are invaluable for *lexicographers* – people who write dictionaries - and for terminologists - people who study the vocabulary of a particular domain.

Obviously, one cannot use a corpus simply as a collection of text to be read like a book – it is far too large. In order to extract the evidence contained in a corpus, linguists

perform operations on it such word frequency, extraction of all occurrences of a particular word or phrase and context, or a search for particular structures or grammatical constructs [Leech and Fligelstone, 1992].

2.2 Tools for corpus linguistics

In this section we will describe some of the computerized tools that have been developed for linguists who use corpora in their studies.

2.2.1 Hector

Hector is a system developed between 1991 and 1993 by the Digital Equipment Corporation and the Oxford University Press [Atkins, 1992]. Its focus is computer-aided lexicography and its purpose is to compile dictionary entries using corpus evidence and to sense-tag the corpus lines that have been used to create the dictionary entries. Atkins says "the computer tools make the work faster, more thorough, more consistent, and infinitely more fun.". Since lexicographers at the Oxford University Press were previously tagging words that appear between 500 and 1000 times in the corpus, manual sense-tagging could be tedious and slow.

The corpus used by Hector contains approximately 17.3 million words of current British English, mostly of written texts but also some spoken language. It has been extensively preprocessed: typographical errors were removed, part of speech tags were added to each word, the structure of the text was marked (paragraphs etc.), the text was parsed, and wordform occurrences and word pair frequencies were computed.

The system uses three monitors. The left-hand monitor shows tools which generate statistics on the text, access reference material (such as on-line dictionaries), and provide a checklist of verb patterns. The centre monitor shows corpus texts and performs operations on them (some of which will be described in the next paragraph). The right-hand monitor shows the dictionary entry editor.

At the heart of Hector are the corpus searching and tagging tools which are controlled from the centre monitor. The word to be searched for (called the "target" word) is entered by the user. The word can be expanded to include all its inflections if desired: the user can choose one or more of typographical variations (cat, Cat, CAT), verb forms (punch, punched, punches, punching etc.), noun forms (dog, dogs, dog's, dogs' etc.) and the forms for some other parts of speech.

The search can be constrained by restricting the part of speech of the target word. For example, the user can search for *bank* as a noun only, which would eliminate occurrences where it is used as a verb (as in to *bank a plane*). The search can also be limited by restricting the sense tag of the word for those occurrences that have already had sense tags added. *Bank* could be restricted to the sense of a *river bank*, eliminating the sense of a *bank* where money is kept.

Once the searching criteria is set, hitting the **search** button produces a type of output, called a *concordance*, showing each line in which the target word occurred, The lines are displayed in the order that they appeared in the corpus but once the search is done, the user can then sort the output alphabetically by the word to the left or right of the target word, by the word form of the target word, by the text type in the corpus, or by the sense tag of the target word. Instead of doing a full search, the user can use the **count** button to simply count the number of occurrences that match the search conditions. This is faster than a full search.

Various utilities are available to save the output, to show a concordance line with complete analysis – wordclass tags and clause analysis data, and to assign sense tags to a selected concordance line.

Hector is a tool designed for a specialized use – the compilation of dictionary entries. It contains several useful tools for analyzing text but its narrow focus limits its use to lexicography or related fields.

2.2.2 The Translator's Workbench Project

The Translator's Workbench Project, described in [Ahmad et al, 1994] [Ahmad and Rogers, 1992a] and [Ahmad and Rogers, 1992b], was a three-year project from 1989-92 funded by the Commission of the European Communities. A second phase of the project ran from 1992-94 with its goal to make the results of the project available to the market place.

The need for a computerized translation assistant is obvious when we examine the growing importance of translation in the European Community. By 1995 the community had nine official languages and the amount of text translated in Western Europe for 1987 alone was estimated at 160 million pages.

The Translator's Workbench is designed to provide machine support for translators in the form of an integrated set of software tools designed to eliminate some of the tedious work of translating. The tools include a multilingual editor, checkers for spelling, style and grammar, access to a machine translation program, and tools for building and accessing term banks. We will focus on the last tool called MATE (Machine Assisted Terminology Elicitation) which was developed at the University of Surrey.

Terminologists are linguists that specialize in identifying, describing, standardizing or making up *terms* for a domain. A *term* is a word, or more than one word, that names a concept. Ahmad et al. say "The common denominator seems to be that a term is a label – usually lexical – in the special language of a specific domain, designating a particular concept in the knowledge of that domain, and arguably less context-dependent with regard to its sense than a general-language word." [Ahmad et al., 1994]. For example, in computer science some terms are *operating system*, *hard disk*, and *binary tree*. In sailing, examples of terms are *tack*, *mainsheet*, and *whisker pole*. The words that comprise the term may have one meaning in general language and a different one in a specialized domain For example, the term *tack* in sailing does not describe a pin-like object used to fasten carpet to a floor but a procedure where a sailboat changes direction by turning its front (or *bow*, another term) across the direction of the wind.

Terms are most often nouns since these describe the physical objects or concepts in a domain, but they may also be verbs, adjectives or adverbs.

Terminologists must find the terms for a particular domain but translators have the added problem of finding appropriate terms in other languages. What MATE in the Translator's Workbench does is to aid in the identificaion of terms from a text using a corpus-based approach.

MATE assists in the identification of terms from a corpus by compiling word lists (ordered by frequency or alphabetically), creating a concordance (an alphabetical list of all words in a text with their contexts, and showing collocations (a list of the co-occurrence of specified terms). It also provides for term storage and retrieval.

Like Hector, the Translator's Workbench is a specialized tool, in this case designed for translators only.

2.2.3 TACT

TACT (Text Analysis Computing Tools) is a collection of 15 programs for MS-DOS or Windows developed at the University of Toronto that is designed to do text-retrieval and analysis on literary works in any language that uses the roman alphabet, or in classical Greek. It is described in [TACT, 1990] and [TACT, 1994].

The first step in using TACT is to optionally tag or mark up a copy of the ASCII text. Tags are marked with <> brackets and may include markers indicating chapters of a novel, scenes of a play, date, location, audience, theme or any other helpful indicators. There are also programs to assist the user in adding tags to all the words in the text, such as a part of speech or a conceptual label.

Once the text is tagged, the program *Makebase* is employed to convert the text into a database. Makebase works best on files of 200k or less – larger files must be segmented and then the resulting databases merged together afterwards. TACT uses the technique of ***full-text indexing***, which is described more fully in section 4.5.1. Briefly, it is a way of

storing pointers to all the words in the text so they can be located rapidly. The advantage of full-text indexing is the speed of retrieval. The disadvantages include the time needed to compile the index and the large amount of space needed to store it.

Various retrieval or analysis programs are available to run on the database. *Usebase* selects words, word patterns or a group of words from the text and displays the results in one of five ways: as whole text, as one of two types of concordance display with different amounts of text shown, as an occurrence distribution graph which shows *where* the word occurred in the text, or as a table of collocates which shows what words occur frequently close to this word.

The program *Collgen* lists repeating fixed phrases and two word *collocations* – pairs of words that frequently occur close to each other. *TACTstat* generates statistics for word length and word frequency, *TACTfreq* makes word lists in alphabetical, reverse alphabetical and descending frequency order and *Anagrams* produces anagrams of words. TACT also provides utility programs for comparing and sorting files.

Because TACT was originally designed to do literary research, it was never intended to be used on a large corpus and does not handle large texts well. It is generally for use on text the size of a single book.

University of Ottawa terminologists used TACT when they conducted an in-depth study of certain words. The concordance feature allowed them to locate all occurrences of a word in a large text quickly so that more time could be spent on the analysis of the word rather than simply finding its occurrences [Meyer, 1995]. These researchers are now using the Text Analyzer instead of TACT for several reasons: a) TACT could not handle large texts, b) it was cumbersome to perform some operations that the Text Analyzer does in one step, and c) the Text Analyzer offers more operations and features than TACT.

2.2.4 Xtract

Xtract is a program developed by Frank Smadja [Smadja, 1993] that attempts to find *collocations* in text. According to M. Benson [M. Benson, 1990], a collocation is "an

arbitrary and recurrent word combination". Collocations occur more often than we would expect to happen by chance. *Prime minister*, *make a decision* and *back up* are all collocations.

Smadja identifies four interesting properties of collocations:

1. Collocations are arbitrary, making them difficult for second language learners and machine translation programs since the collocation must be learned or translated as a whole and its meaning may be different from the individual meanings of the words put together. *Shepherd's pie* is a dish made of meat with mashed potatoes on top and is neither a pie, nor a food exclusive to shepherds. To add to the confusion, *shepherd's pie* translates to a different collocation in French: *pâté chinois*.

2. Collocations are domain-dependent and in a particular domain may have a different meaning than in general language or in another domain. A *hot key* in a computer application is a combination of keystrokes used to invoke a function faster than by picking it from a menu. This usage is restricted to the area of computers.

3. Collocations are recurrent; they are repeated often in a particular context or domain. This means that they are not one-time exceptions.

4. Collocations are cohesive clusters of words; often the presence of one word of a collocation implies the rest of it. English speakers naturally would fill in *"to ... a bicycle"* with *ride* but would fill in *"to ... an airplane"* with *fly*.

Because of these properties of collocations, they have particular statistical distributions. Smadja gives the example that the probability that any two adjacent words in a text will be *red herring* is greater than the probability of *red* times the probability of *herring*. We can, therefore, use statistical methods to find collocations [Smadja, 1993].

The input to the program is a single search word for which we want to find collocations. Then Xtract proceeds in three stages. The first stage consists of extracting *significant* pairs of words (called *bigrams*), each consisting of the search word and a word that occurs within 5 words on either side. *Significant* means that these two words appear together sig-

nificantly more often than expected by chance and that the words appear together in a relatively rigid way. Statistical methods are used to measure the significance of each bigram.

In stage two Xtract uses the bigrams found in stage one to find longer collocations that include these bigrams. For example, *air traffic* would be replaced by *air traffic controller.*

Stage three attempts to add syntactic information to the collocations. Smadja gives the example of the collocation *make decision.* It is not enough, he says, to know that *make* goes with *decision* – it is also important to know that *make* is the verb with *decision* as its direct object. If a consistent pattern like this is not found, the collocation is rejected. The output of Xtract is a set of sentences for each collocation with syntactic labels.

Why is it interesting to find collocations? First, a collocation may be the name of a concept such as *Cocker Spaniel, prime minister* or *gross national product* that we want to identify in a text. Secondly, a collocation that consists of a noun and a verb may indicate either a subject-verb or verb-object phrase that customarily goes together. Some subject-verb collocations are: *weatherman forecasts...*, *professor teaches...*, and *car breaks down.* Examples of verb-object collocations are: *kill a process*, *make a decision*, and *drive a car.* This kind of information is necessary for translators and is important also for knowledge engineers trying to do a conceptual analysis of a domain.

2.3 Knowledge management systems

Now we will look at research in computer science. Knowledge management systems are computer programs that store, organize and retrieve knowledge. They attempt to handle knowledge in a way that their developers believe is more compact, informative and convenient than conventional knowledge storage forms such as books. Their knowledge is usually stored in a *knowledge base*, a structured repository of small nuggets of information. [Skuce and Lethbridge, 1995 to appear]

A knowledge base is supposed to be superior because it stores just the facts, not sentences as in a book. The structure of a hierarchical knowledge base also quickly indi-

cates the structure of the knowledge represented (see figure 2.1 and 2.2). A knowledge base may also be designed so that its contents may be accessed and used by another computer program, such as an expert system – a knowledge-based program that attempts to solve problems in a limited domain.

The following sections examine two knowledge management systems that take different approaches.

2.3.1 CYC

CYC, described in [Lenat et al., 1990] and [Lenat and Guha, 1990] is an enormous project that has been under development since 1984 under the direction of Douglas Lenat. Its purpose is to build an extremely large knowledge base that when complete, will contain on the order of 10^8 axioms, or statements of fact.

CYC was developed to answer a problem that Lenat sees occurring with current artificial intelligence systems. He believes that the emphasis of research has been placed on procedures for using knowledge, without first gathering the knowledge needed by these procedures. He sees expert systems as being very brittle because they cannot handle situations that are out of the ordinary. For example, an old car was diagnosed by a skin disease diagnostic program as having measles because the "patient" has reddish-brown spots on its body. Lenat says that brittleness is the result of not enough common sense, or background knowledge. Common sense would allow this program to realize that an old car is not a "patient" because it is not alive. Humans can easily cope with situations like this because we can use our knowledge of similar cases and propagate the differences to the new situation.

The CYC project, therefore, was designed to build a comprehensive knowledge base of general knowledge about the world that could then be used by other programs. This ambitious project has now been under way for 11 years. One of the reasons that it is taking so long, apart from the large quantity of knowledge to be stored, is that up to 1989 the knowledge was entered manually. After this date, CYC itself was able to do some internal inferencing to find connections between the pieces of knowledge it already contained.

Eventually it was hoped that CYC would do more and more of its own knowledge acquisition and that the role of humans would shift to that of teacher, recommending what to read next and explaining difficult passages.

CYC has two user interface tools for examining and editing the knowledge base: a "symbolic spread sheet" frame editor and a spatial "museum room" editor, but they are strictly for expert users. Another user interface was tried initially but was "a dismal failure" according to Lenat. This graphical node and link editor worked nicely until there were more than 5,000 nodes in the system when it became too complicated and cluttered to read.

It will be interesting to see how CYC progresses over the next few years and whether it eventually reaches its goals.

2.3.2 CODE4

CODE4 is a knowledge management system developed at the University of Ottawa and is described in [Skuce and Lethbridge, 1995 to appear] and [Meyer, Skuce, Bowker and Eck, 1992]. Like CYC, it is designed to store knowledge but its emphasis is on helping *humans* organize, define, understand and display knowledge, and it works on a much smaller scale. Skuce and Lethbridge see the need for "knowledge-based systems which primarily act as *amplifiers of human intelligence* rather than systems designed to run autonomously..." Unlike CYC, CODE4 can be used by ordinary people to build their own knowledge bases from scratch.

CODE4 is a general purpose tool that can be used in a wide variety of situations. It has been used by software engineers to plan and clarify software designs, by terminologists to build knowledge bases that can be used as term banks but which contain more information than a traditional term bank, and by students to aid in the understanding of difficult subjects. Its main strengths lie in the structuring of knowledge that makes it easy to understand and in the sophisticated display facilities that allow for the presentation of knowledge in different ways.

CODE4 provides for a hierarchical structure of *concepts* with associated *properties* and *values*. The concept-property-value triple is called a *statement* – it represents a statement of one fact. For example, CODE4 might contain a statement about the concept *horse* with the property *number of legs* which has the value *four*. Concepts which are lower down in the hierarchy (more specific) inherit properties from the concepts higher up (more general). The concept *horse* would inherit properties from the more general concept *domestic animal* and *horse* would itself pass properties down to its subconcepts such as *race horse* and *draft horse*. CODE4 also allows for multiple inheritance in both its concept and property hierarchies: a *pet fish* may inherit from both *pet* and *fish*.

A concept may have any number of properties which may themselves be arranged in a hierarchy. The concept *bicycle* could have a property *parts* which itself has the subproperties *frame, wheels,* and *gears*.

CODE4 separates the ideas of a *concept* and the *term* or *terms* used to denote it. A concept may have more than one term: *sofa, couch* and *chesterfield* might all be terms for the same concept, or the knowledge base might contain terms in several languages. As well, more than one concept may have the same term: a concept in a hierarchy of financial institutions may use the term *bank*, while in the same knowledge base the same term *bank* might appear as a geographical formation.

CODE4 has several levels of user interaction modes from beginner to developer, each presenting more facilities than the previous one. This allows a naive user to see only the basic operations needed while allowing a more sophisticated user to make use of the more complex operations. The level of formality is also optional and the user can store knowledge in an informal or more formally structured way if desired.

A knowledge base in CODE4 may be viewed in a graphical or textual format. Figure 2.1 shows a graphical display of a knowledge base about people. The left side shows the concept hierarchy with the most general concept *person* on the left. The links between nodes show inheritance – for example, *James T. Kirk* inherits both from *adult* and *fictional person*. The concept *teacher* is selected so on the right side of the window in the

upper pane is the property hierarchy for the concept *teacher* with the property *job* selected. In the lower pane is the area for editing the property value.

Figure 2.2 shows the same knowledge base as a textual display.

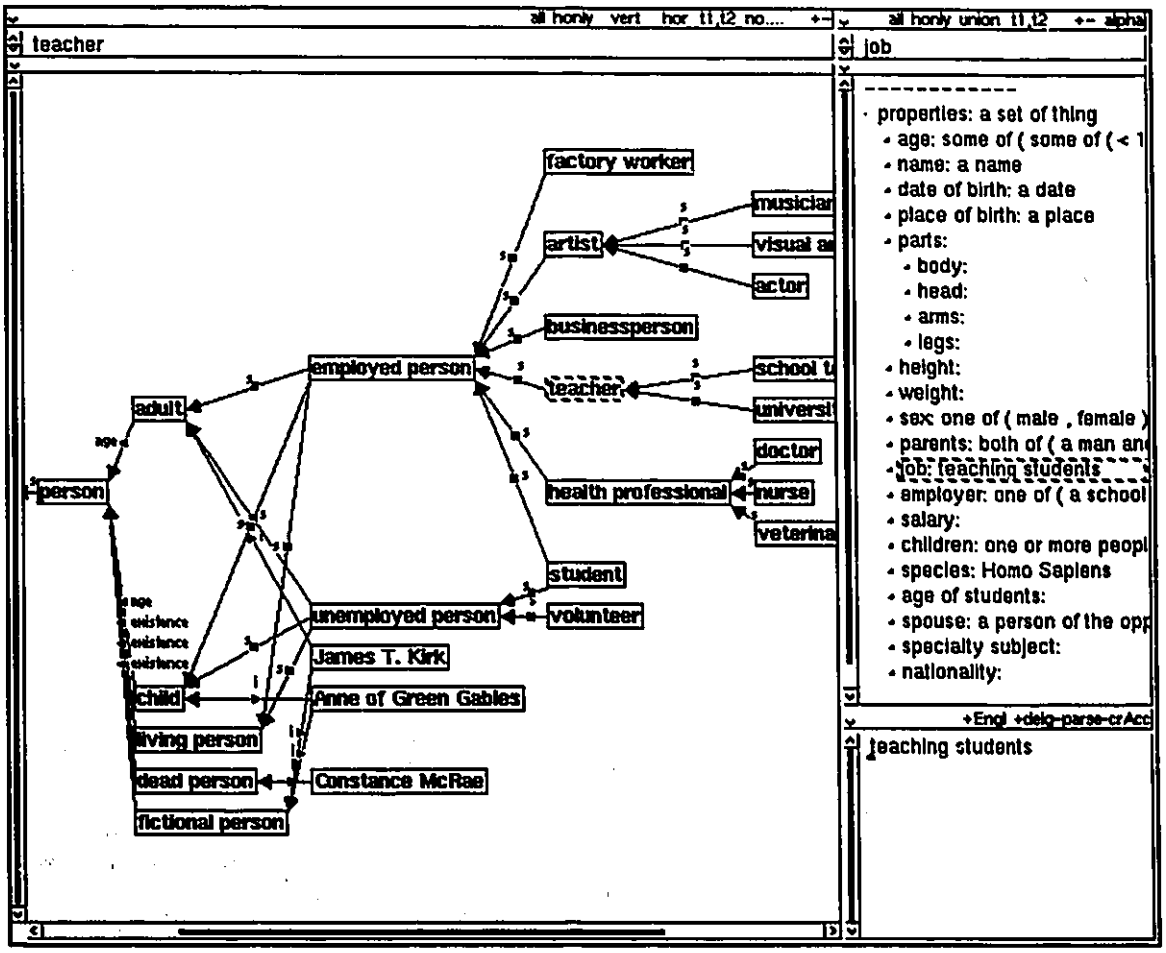

Figure 2.1 A graphical display of a knowledge base in CODE4

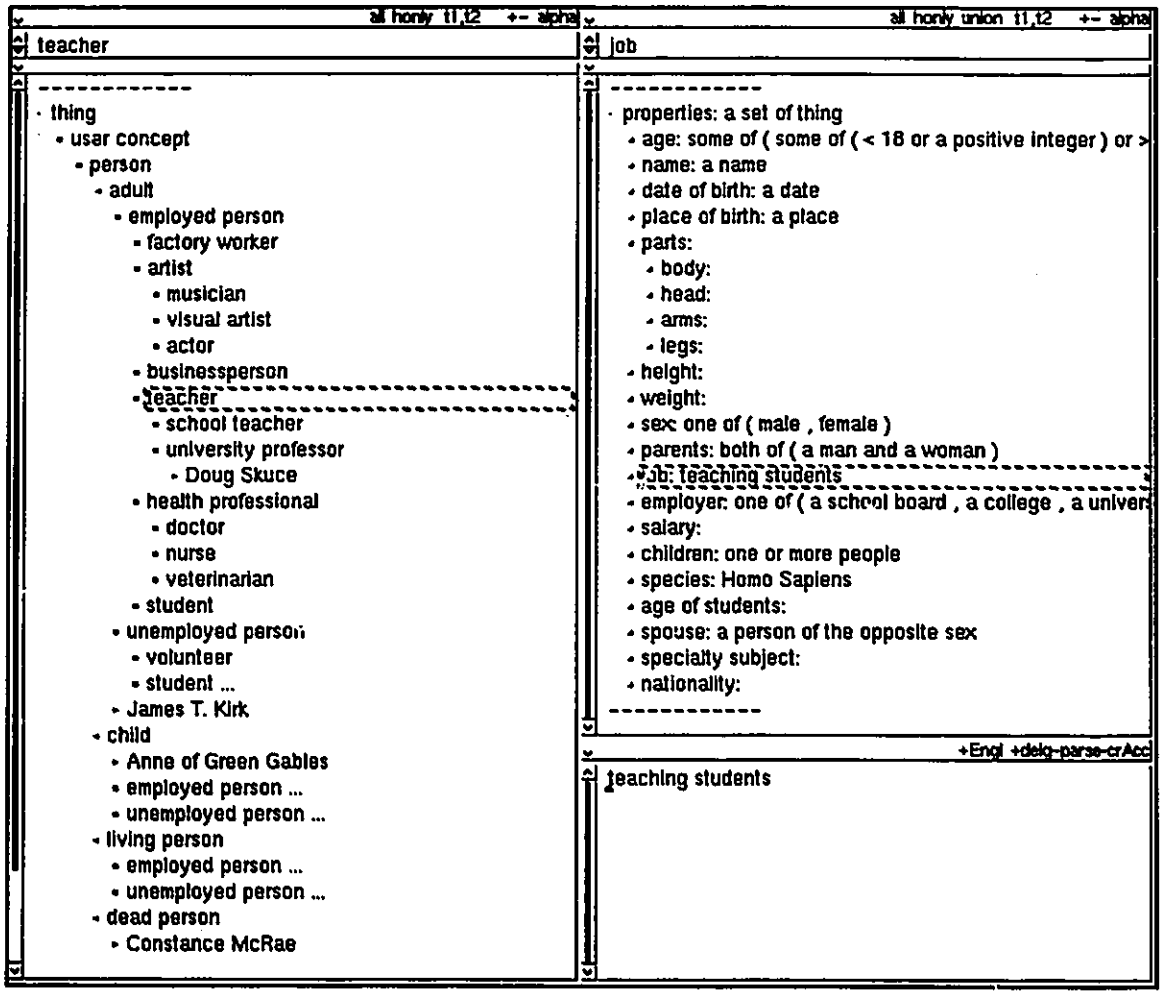

Figure 2.2 A textual display of a knowledge base in CODE4

One of the drawbacks of CODE4, as for CYC, is that the knowledge used to construct a knowledge base has to be extracted from a book or a domain expert's brain and entered into the knowledge base manually. Unfortunately, experts do not have a lot of time to sit around transferring knowledge into a knowledge base or even to be interviewed by knowledge engineers; because they are experts, they are usually busy solving problems requiring their expertise. This leaves texts as perhaps the main knowledge source for knowledge base builders. Finding knowledge in a text and entering it into a knowledge base is a

very labour-intensive process. The knowledge base builder must first locate the main concepts in the text, try to define them and understand their relationships with other concepts, and attempt to discover important properties of the concepts. This is such a large problem that transferring knowledge into a knowledge base has become known as the *knowledge acquisition bottleneck*. The Text Analyzer aims at helping to "unstop" this bottleneck.

2.4 Bridging the gap between linguistics and knowledge management systems

In the last few years researchers have started to recognize the contributions that linguistics, especially corpus linguistics, could make in the field of knowledge engineering and vice versa. A few systems have been built that incorporate tools from both domains. We will look at two such tools here.

2.4.1 LEXTER

LEXTER is a software package for extracting terminology that was developed at the Research and Development Division of Electricité de France (the French Electricity Board). LEXTER is described in [Bourigault, 1995], [Aussenac-Gilles et al., 1995], [Bourigault, 1994], [Bourigault and Lepine, 1994].

Aussenac-Gilles and Bourigault propose its use in knowledge acquisition. In [Aussenac-Gilles et al., 1995] the authors state, "For some time now, some terminologists have been claiming that knowledge acquisition should take better account of terminology; more rarely, knowledge engineers have encouraged their colleagues to pay attention to results of research in terminology." They believe that knowledge engineering mistakenly tends to concentrate on problem-solving knowledge rather than domain knowledge. They propose that terminological knowledge bases (knowledge bases containing information about the terms of a domain) would be a useful reference for knowledge engineers constructing knowledge-based systems.

Unlike knowledge acquisition used in the creation of knowledge-based systems such as expert systems, knowledge acquisition for terminology is not always guided by a

specific application. For a terminologist, the corpus is a source of information and is also a reference. The results must remain faithful to the corpus and will remain a static record of the knowledge contained in that corpus.

A knowledge engineer with a specific application in mind can locate background information in a terminological knowledge base, but then she must also perform a complementary *conceptual* analysis to build a knowledge base for a specific knowledge-based system. While a terminologist is concerned mostly with words, a knowledge engineer is concerned also with the ideas that the words represent.

According to Aussenac-Gilles and her colleagues, few knowledge acquisition methods make use of the linguistic level of words and texts or of domain terminology. (CODE4 is an exception as it labels each *concept* of the knowledge with one or more *terms*, which are themselves treated as full-fledged objects). As a result, confusion often exists between the terms of the domain and the names labeling the conceptual structures in a knowledge base. She says "To label the conceptual structures, the knowledge engineer is free to define new terms or to combine several domain terms, which may never appear as such in texts". This can cause a problem when the knowledge base is used as a basis for discussion with domain experts – it is better if the original domain terms are used.

Aussenac-Gilles cautions that "...the use of a terminological scanning tool like LEXTER is justified only if the documentary corpus is large enough to prevent any exhaustive analysis of it directly by the knowledge engineer."

Using a corpus of English or French texts, LEXTER first does local syntactic parsing based on surface patterns in order to extract noun phrases which are likely to be terms. These compound terms are linked together with other terms which use the same constituent words to form a network. The user then scans through the proposed terms, eliminating some, finding synonyms, picking out generic terms that represent main concepts and adding extra links as needed between nodes. The result is a terminological hypertext web which has been validated and enriched by the knowledge engineer. This terminological hypertext web is then used as a searching tool to investigate related concepts.

The authors in [Aussenac-Gilles et al., 1995] describe a planned experiment to combine a terminology extraction tool (LEXTER) with a knowledge acquisition tool (MACAO). They wanted to study the contributions that terminology can make to knowledge acquisition and, in particular, how knowledge acquisition can benefit from (a) terminological knowledge bases, (b) tools used by terminologists, and (c) methods used by terminologists.

When using MACAO, the knowledge engineer is asked to identify the vocabulary of the domain. The resulting words are called *terms* and the *lexicon* is the list of terms. The knowledge engineer is supposed to use the terms in the lexicon to label the conceptual structures. If a new label is invented for a conceptual structure, it is added to the lexicon as a term. Aussenac-Gilles found this could lead to confusion between terms that appear as domain vocabulary and terms invented to label conceptual structures. Also, the knowledge engineer is given little guidance on how to identify the domain vocabulary and the task is often time-consuming.

Aussenac-Gilles and her colleagues believe that knowledge including technical documents and human know-how could be gathered into a network of structured but informal knowledge that they call a *knowledge bank*. It would contain both terminological and conceptual knowledge and would be useful for a wide range of systems.

2.4.2 The Knowledge Acquisition Workbench

The Knowledge Acquisition Workbench (KAWB) [Mikeev and Finch, 1995] is a knowledge engineering workbench that uses methods from computational linguistics, information retrieval and knowledge engineering to extract knowledge from texts. Mikeev and Finch call this process *corpus-based knowledge engineering*.

Mikeev and Finch note that a major bottleneck in using many knowledge-based systems in a new domain is the acquisition of background knowledge of the new area and that since much of expert knowledge about almost any topic is encoded in texts, automatically channeling some of that knowledge out of the text and into the knowledge base would

greatly aid knowledge acquisition. They believe that existing techniques from the field of expert systems are not linguistically sophisticated enough. According to Mikeev and Finch, other tools such as KRITON [Diederich et al., 1987] and SHELLEY [Anjewierden et al., 1992], which use hypertext to make texts mouse-sensitive, are suitable only for small texts. The next step, they propose, is to use linguistic and statistical methods to uncover clues in the text that indicate important semantic features of the domain.

KAWB, which was partly implemented in 1994, incorporates two modules. The first one, the data extraction module, includes several components and tools.

The word class identification component tags each word in the text with its part of speech, finds root forms of inflected words (*works*, *worked* and *working*, are inflected forms of the root *work*), groups structurally related sequences of words together (for example, *a can of worms*), and finds subject-predicate relations. The term clustering tool performs semantic clustering on terms based on the idea that words are similar if they are used in similar ways. A statistically based similarity measure between words is computed and a hierarchical single link clustering, called a dendrogram, is produced. The collocator finds collocations in the text and the generalizer attempts to generalize these using the dendrograms produced by the term clustering tool and the external lexical database WordNet. For example, given a number of collocations about *specific* diseases of *particular* parts of the body, it can find the general pattern that indicates that diseases affect body parts.

The second module is the analysis and refinement module whose purpose is to find and refine structural generalities. It takes patterns representing hypotheses of the knowledge engineer, tries to match them in the text, groups and generalizes the results found and presents them to the user. Creating a hypothesis consists of filling in generic conceptual structures (thematic case frames) with the specifics for this domain.

Two aspects of Mikeev and Finch's research are interesting in relation to our work: 1. the idea of corpus-based knowledge engineering and 2. the concept of one workbench incorporating many different tools. These are key ideas in the development of our Text Analyzer which is described in the next chapter.

3 The Text Analyzer

In this chapter we introduce the Text Analyzer and examine each of its operations in detail.

3.1 Introduction

We designed the Text Analyzer as a multi-purpose tool that brings together an assortment of analysis techniques from linguistics (especially corpus linguistics) to assist both knowledge engineers and linguists in their work of extracting knowledge from documents. Many other tools may use one or two analysis methods but we have attempted to provide a wider spectrum of techniques all together in one tool. The Text Analyzer is also connected to the CODE4 Knowledge Management System to allow for the easy construction of knowledge bases from the knowledge found in the document.

Figure 3.1 shows the procedure for using the Text Analyzer. The text must first be preprocessed so that each sentence appears on a separate line in the file. Then the text can be tagged with part of speech tags, and compounds (like *prime minister)* marked as referring to a single concept. Tagging and grouping compounds are strictly optional but can be useful when performing other analysis operations. The statistics operation provides information about the words and sentences in the text file.

In the analysis phase the usual first step is to identify the terms in the text using the frequency operations. Then the user analyzes one term at a time by performing various operations on each term. There are a number of operations with several options each to choose from. The figure shows one possible sequence of operations but the user can run whichever sequence of operations she wishes in whatever order is preferred. This forms a basic premise of the Text Analyzer – that there are many different methods of text analysis and the user should not be restricted to any particular one. She can pick and choose from the operations and perform them in any sequence. Some of the operations overlap in the results they produce but the methods used are different – this is an interesting test of the methods as well as an opportunity to gather as much knowledge as possible.

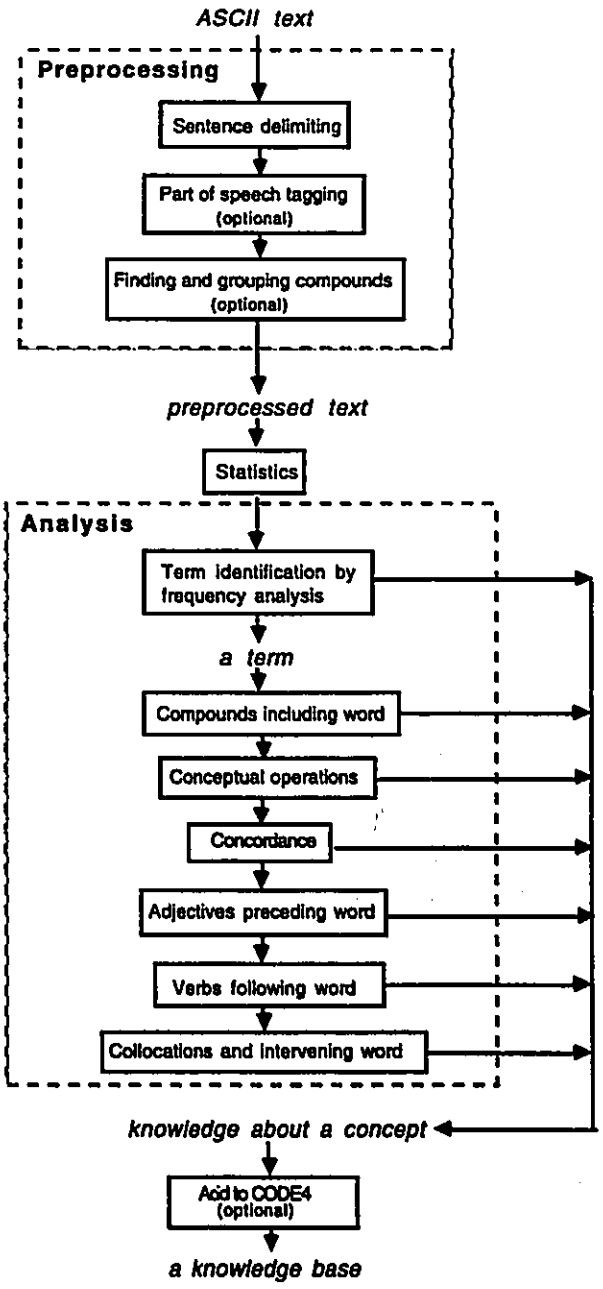

Figure 3.1 Flow chart of the procedure for using the Text Analyzer

The output of the Text Analyzer may be used directly – for example, to verify an existing text or to understand the ideas contained in it. The output may also be used to build a CODE4 knowledge base which more concisely and permanently stores the knowledge gleaned from the text, and which acts as a quick reference for this knowledge later on.

The Text Analyzer was implemented in Icon, Smalltalk and C to run under Unix.

In the next sections we will describe in more detail how the Text Analyzer is used and how it works.

3.2 The user interface

Figure 3.2 shows the Text Analyzer's user interface. The top window displays the text of the document being analyzed, with the name of the file above it. The text can be scrolled up and down with the scrollbar at the left of the window. Clicking on the middle button of the mouse brings up a menu of operations that can be performed on the document.

The large middle window shows the output of the operations. The output can be edited and saved to a file and can also be used as input for other operations. Clicking on the middle button of the mouse in this window brings up a different menu of operations.

At the bottom of the Text Analyzer is the area for assembling knowledge to add to CODE4. The buttons at the left allow for transferring of text from the middle window into the subject, property and value lines. Text can also be copied and pasted, or simply typed into these lines. The **add to KB** button adds the assembled information to a knowledge base.

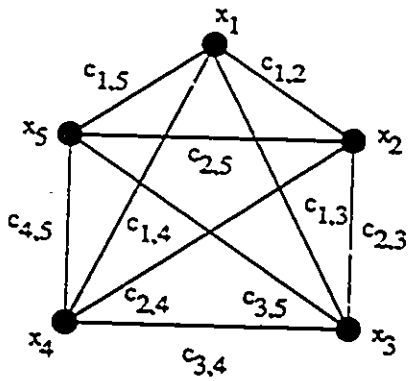

Figure 4.5: The graph obtained when we shrink our family of curves.

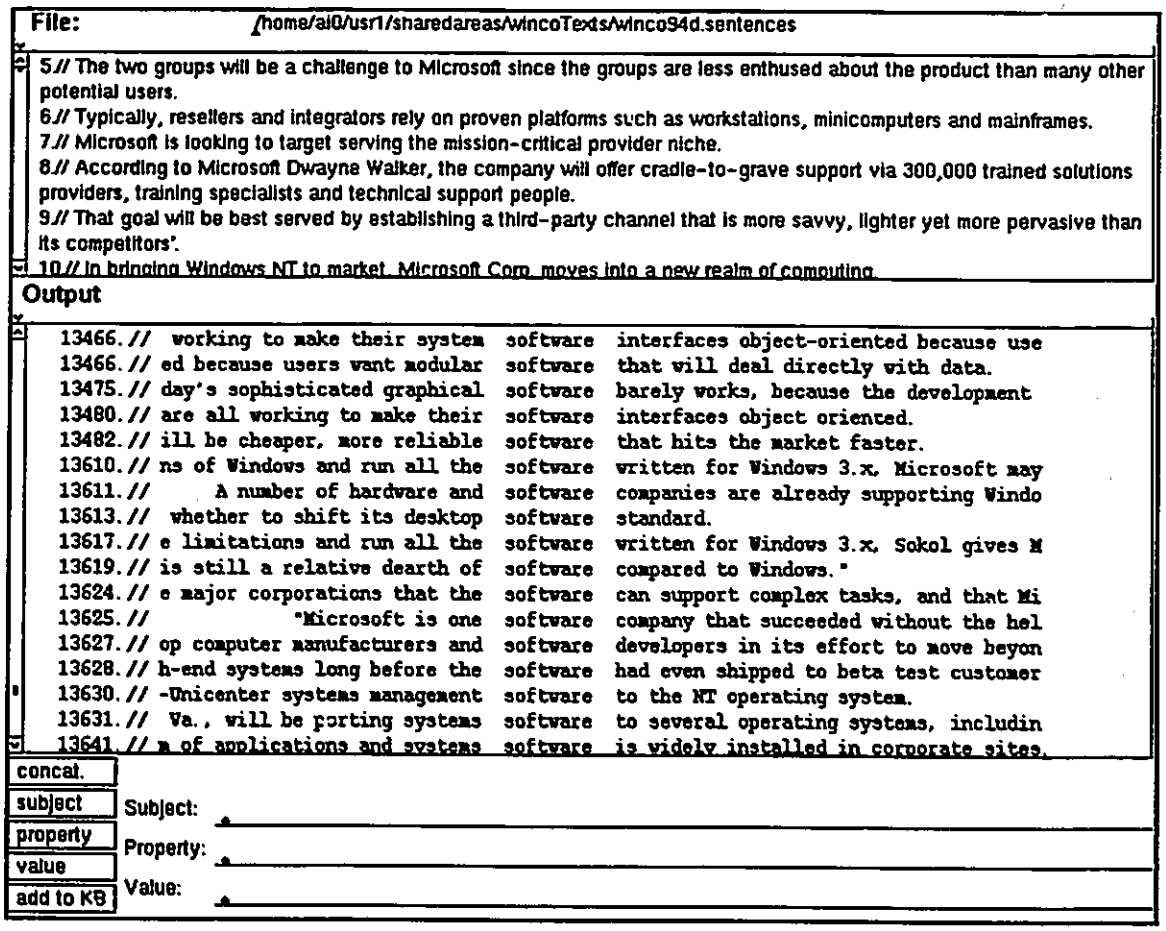

Figure 3.2 The Text Analyzer's user interface

3.3 Operations

This section describes the various operations that the Text Analyzer performs.

3.3.1 Preprocessing

Preprocessing prepares a text in ASCII form for use by the Text Analyzer. The first preprocessing step, *sentence delimiting*, is mandatory. The other two types, *part of speech tagging* and *finding and grouping compounds* are optional but useful.

3.3.1.1 Sentence delimiting

Before any work can be done on the text it must be divided up into sentences. Each sentence is numbered, making it is easy to refer to a particular sentence in the text.

Finding sentence boundaries is not as easy as it might seem. Simply looking for a period at the end of a sentence can lead to problems – an abbreviation containing periods within a sentence (such as U.S.A.) can lead to fragmentation of the sentence whereas headings and titles which do not end in a period lead to concatenation of what should be several sentences. The sentence delimiter program uses various heuristics to try and overcome these problems but sentence delimiting remains somewhat inaccurate. On the whole it seems better to concatenate together multiple sentences than to cut sentences up into pieces since many of the processing operations work on the sentence level and therefore no information would actually be lost. If a sentence was fragmented, an operation that examines a single sentence might miss valuable information given only a part of the sentence.

To process a line of text from the file, the program examines each word in turn to see if the end of the sentence has been found. First we will examine the case when the last character of a word is a period. This may mark the end of a sentence or it may be an initial (*L. L. Bean*), or an ellipsis (*and so on...*) or an abbreviation (*U.S.A.*). Possible abbreviations are checked against the list of abbreviations contained in Appendix C.

If the last character in the word is not a period, it still marks the end of a sentence if it is a question mark or exclamation mark. If the last character is a quotation mark the second last character must also be checked. (In *He cried "Jump."* the second quotation mark is at the end of the sentence because the second last character is a period, whereas in *"Jump" he cried.* the second quotation mark is not at the end of the sentence because the second last character is not a sentence-ending character, such as a period). If the last character in the word is a closing bracket we must check to see if the sentence started with a bracket, in which case this marks the end of the sentence, or if the sentence did not start with a bracket, in which case this is a bracketed section in the middle of a sentence. If the

bracketed section was at the end of the sentence, the last character would be a period rather than a bracket.

Titles, headings, lists and other structures that do not contain normal sentence punctuation still present problems and still tend to get lumped together as long sentences at the present.

The following paragraph is from a small corpus (approximately 173,000 words) about optical storage media made up of articles published in 1994 and taken from Computer Select [Computer Select, 1994], a monthly CD-ROM publication that contains articles and extracts from over 140 computer industry publications:

```
It's hard to argue with the convenience of catalog
shopping, but there are drawbacks: Catalogs waste a lot of
paper, they're expensive to produce and mail, and you can't
try out stuff before you buy. If several vendors, Apple
included, have their way, CD-ROM technology is the answer. In
these schemes, software is encrypted on the CD-ROM. To
purchase an application, you phone a toll-free number and
fork over your credit-card number; in return you receive a
code that lets you unlock the program, after which you can
download it to your hard disk.
```

Figure 3.3 A sample text

After putting the text through the sentence delimiter, this fragment is divided into sentences as follows:

```
4.// It's hard to argue with the convenience of catalog
shopping, but there are drawbacks: Catalogs waste a lot of
paper, they're expensive to produce and mail, and you can't
try out stuff before you buy.

5.// If several vendors, Apple included, have their way, CD-
ROM technology is the answer.

6.// In these schemes, software is encrypted on the CD-ROM.
```

```
7.// To purchase an application, you phone a toll-free number
and fork over your credit-card number; in return you receive
a code that lets you unlock the program, after which you can
download it to your hard disk.
```

Figure 3.4 The sample text after sentence delimiting

3.3.1.2 Part of speech tagging

During this optional preprocessing step, each word in the text is marked with its part of speech (noun, verb, adjective etc.) using the rule-based part of speech tagger written by Eric Brill. (See Appendix A for the list of Penn Treebank part of speech tags used by the tagger). Knowing the part of speech of each word is very useful for some operations. For example, it allows the user to compile a word list of only the nouns used in a text, or to find what verbs occur with what nouns.

Various methods have been used for automatic part of speech tagging that fall under two main categories: taggers based on a statistical approach, and rule-based taggers. In the *statistical* approach, using a small manually tagged text as an training example, a model is built based on the statistical evidence of the training text. Once the tagger is trained, new text can be tagged by assigning it the tag sequences given the highest probability by the model. *Rule-based* part of speech taggers depend on a set of rules that assign a tag to a word based on the surrounding words. These were the first kind of taggers to be explored. In the early 1960's part of speech taggers were created with totally hand-crafted rules, sometimes written with the aid of a corpus. Statistical part of speech taggers have generally been more successful because they had better accuracy and were more robust than rule-based taggers. Rule-based taggers, however, need to store much less information, are easy to modify, use a small set of rules that are easily understandable by humans and are more portable to other languages.

The part of speech tagger used by the Text Analyzer is a rule-based tagger created by Eric Brill that uses transformation-based error-driven learning to automatically acquire its own set of rules from a training text. It is described in [Brill, 1992], [Brill and Marcus,

1992], [Brill, 1993] and [Brill, 1994]. It has an accuracy comparable to statistical taggers: once the tagger has been trained it can be used on unfamiliar text with an accuracy of about 96%. Apart from a few mistagged words, we have found that this rate of accuracy is quite sufficient for our purposes where we mainly need to identify nouns, verbs and adjectives. At first we thought it would be necessary to retrain the tagger on our corpus but it performed well enough off-the-shelf for our work.

When run through the part of speech tagger, the sentences about optical storage media that we saw in the previous section become:

```
4.// It/PRP 's/VBZ hard/JJ to/TO argue/VB with/IN the/DT
convenience/NN of/IN catalog/NN shopping/NN ,/, but/CC
there/EX are/VBP drawbacks/NNS :/: Catalogs/NNP waste/NN a/DT
lot/NN of/IN paper/NN ,/, they/PRP 're/VBP expensive/JJ to/TO
produce/VB and/CC mail/NN ,/, and/CC you/PRP can/MD 't/CD
try/NN out/IN stuff/NN before/IN you/PRP buy/VBP ./.

5.// If/IN several/JJ vendors/NNS ,/, Apple/NNP included/VBP
,/, have/VBP their/PRP$ way/NN ,/, CD-ROM/NNP technology/NN
is/VBZ the/DT answer/NN ./.

6.// In/IN these/DT schemes/NNS ,/, software/NN is/VBZ
encrypted/VBZ on/IN the/DT CD-ROM/NNP ./.

7.// To/TO purchase/VB an/DT application/NN ,/, you/PRP
phone/VBP a/DT toll-free/JJ number/NN and/CC fork/NN over/IN
your/PRP$ credit-card/NN number/NN ;/: in/IN return/NN
you/PRP receive/VBP a/DT code/NN that/WDT lets/VBZ you/PRP
unlock/VBP the/DT program/NN ,/, after/IN which/WDT you/PRP
can/MD download/VB it/PRP to/TO your/PRP$ hard/JJ disk/NN ./.
```

Figure 3.5 The sample text after part of speech tagging

It can be seen that the tagger does not correctly tag all the words in this example. In sentence 4 it has tagged *waste* as a noun (NN) instead of a verb and in sentence 7 it also has tagged *fork* as a noun (NN) instead of a verb. More can be done with the tagger to achieve better accuracy, such as training it on the type of text to be used and creating an individual lexicon containing words from the domain to be analyzed. We have not yet

investigated these options because we have not found the tagging errors to be much of a problem in the work we have done so far.

3.3.1.3 Finding and grouping compound nouns

Once the text has been tagged with parts of speech, the last preprocessing step, finding and grouping compound nouns, can be performed if desired. A compound noun is a series of nouns and adjectives (ending with a noun) that denote one concept, such as *hard disk, car door hinge*, or *chimney cleaner*. These concepts are often specialized types of objects: a *car door hinge* is a type of *hinge* and a *chimney cleaner* is a type of *cleaner*. It is important to mark compound nouns as single entities in the text so they can be located and treated as if they were single words. We don't want to have to look through all the references to different kinds of *cleaners* when all we want to see is references to *chimney cleaners*.

The operation that finds possible compound nouns looks for all sequences of consecutive nouns or adjectives, ending with a noun, that are at least two words long. It then outputs a list of them for the user to examine as well as the number of times each one occurs. The output can be displayed as an alphabetical list or sorted by frequency of occurrence with the user specifying the minimum frequency to be displayed. No attempt is made at present to combine singulars and plurals of the same compound or to amalgamate the same compounds starting with upper and lower case letters – features that would be useful.

The following list shows the 20 most frequent suggested compound nouns found in our example text about optical storage media.

```
1. Product Announcement          106
2. Brief Article)                103
3. CD-ROM drives                 83
4. CD-ROM drive                  48
5. hard disk                     36
6. optical storage               36
7. List Price                    34
8. Software Review               32
9. CD ROM                        32
```

```
10. hard drive                  24
11. video clips                 23
12. hard drives                 23
13. Pinnacle Micro              23
14. storage capacity            21
15. audio CDs                   21
16. Media Vision                21
17. Hardware Review             20
18. Voyager Company             20
19. CD-ROM technology           19
20. optical disks               19
```

Figure 3.6 The 20 most frequent suggested compound nouns

Not all of these suggested compound nouns are really compound nouns. *Product Announcement* and *Brief Article*, are merely phrases that occur frequently in the text. Some of the other suggested phrases are compound nouns but are not important in the domain that the user is studying. The compound *San Jose* occurs 19 times in this corpus but if we are examining the domain of optical storage, it is probably not going to be of interest to us. Several singulars and plurals should be combined into one term, thus changing their total frequencies: *CD-ROM drive* and *CD-ROM drives*, and *hard drive* and *hard drives*. We also note the variations in spelling of CD-ROM and CD ROM.

Because the suggested compounds may not all be valid, the user is given an opportunity to manually edit the list and delete unwanted compounds. Additional compounds may be added if they have not been found due to the part of speech tagger mislabeling nouns as other parts of speech. (For example, in the compound *operating system*, the tagger sometimes tags *operating* as a verb instead of a noun).

Once the list is complete, a second operation uses the list to group together all occurrences of these compounds in the text. An equals sign is placed between the words and a single noun tag at the end. The sentence:

```
1.// The/DT e-mail/NN address/NN is/VB in/IN my/PRP home/NN
directory/NN ./.
```

would become:

```
1.// The/DT e-mail=address/NN is/VB in/NN my/PRP
home=directory/NN ./.
```

This completes the discussion on the preprocessing operations. The next sections will examine operations that are perfonned on the preprocessed text.

3.3.2 Statistics

The statistics operation examines the input text and generates statistics about it including a total word count, a word count by word length, a sentence count, and a sentence count by sentence length. It also displays the number of sentences containing words starting with each letter of the alphabet, broken down into words of less than or greater than or equal to five letters. This last statistic is a by-product of a method for fast searching, described in detail in section 4.4.2.

In the optical storage file the statistics operation finds there are 9990 sentences and a total of 172,937 words. The most common sentence length is 4 words (There are many short headers that count as sentences). The most common word length is 3 letters.

3.3.3 Frequency operations and term identification

One of the first tasks in the analysis of a text or corpus, whether the goal is to understand or verify the text, build a knowledge base about it or study its terminology, is to find the *terms* used in the text. As described in more detail in section 2.2.2, a term is a label for a concept in a domain. It is important to discover the terms of a domain since they are the names of the most important concepts in the domain. Identifying the terms in a text gets us a long way towards finding the important concepts in the text.

Discovering the terms used in a particular text may also allow us to find inconsistencies in the terms used between different texts or even within one text. It is important that everyone using the texts or knowledge base use the same terms for the same concepts, or at least, understand that more than one term is being used for the same concept. If it is found

that the same term is being used for different concepts (which may lead to unnecessary confusion) it may be possible to invent a new term for one of the concepts instead.

A lot of work has been done in the area of automatic term identification [Ahmad and Rogers, 1992a], [Ahmad et al , 1994]. We have not attempted to implement a term-finding algorithm as such but instead use a simple but relatively effective method.

How do we locate the terms in a text using the Text Analyzer? One way is to use the compound noun finding operation but this finds only groups of nouns and adjectives and terms may contain other parts of speech. Another way is to look at word frequency.

Given a large enough corpus, if we look at the frequencies of all the words we will find that no matter which corpus we examine, the rank of the most common 50 or so words is approximately the same [Ahmad et al., 1994]. The most frequent word is usually *the*, followed by *of*, and various combinations of *and*, *a*, *to*, *in* etc. [Francis and Kucera, 1982]. If, however, we look at the frequencies of words in a corpus containing texts from a single domain, we will note that within the top ranking words appear some specialized words from the domain. These words are most often the words chosen as terms. They appear high up in the frequency list simply because they describe concepts that are important in the domain and which are, therefore, mentioned often in the text.

The Text Analyzer offers various frequency operations that can be used to identify terms as well as to simply gather statistics on word frequency. The simplest operation is to find the number of occurrences of each word used in the text. The results can be ordered alphabetically, or by frequency with the most common words at the top. A cutoff value can be specified so that only words that appear more frequently than the cutoff will be shown. This can greatly reduce the size of the output. Words starting with a capital letter are not merged with the same word starting with a lower case letter, an improvement to the program that has not yet been implemented.

When using this operation as an aid for term identification, the user can specify that any word that is contained in a list of stop words not be shown. The stop words list is printed in Appendix B and was compiled from the 100 or so more frequent words from the

Brown Corpus plus some other words, such as months of the year, that occurred in the corpus we were using (this list is user tailorable). By eliminating these words, we find that the most common words left in the frequency list are probably terms.

The technique of using the frequency operation for term identification has its limitations though. A word that appears infrequently in the text may also be an important term but would not be spotted easily with this technique.

The following list show the top 25 words by frequency in our small example corpus containing four texts about optical storage media:

1.	the	6925
2.	and	4539
3.	of	3857
4.	to	3826
5.	a	3747
6.	for	2038
7.	is	1999
8.	in	1691
9.	The	1570
10.	that	1277
11.	with	1275
12.	on	1254
13.	you	1094
14.	CD-ROM	1006
15.	are	995
16.	as	923
17.	drive	849
18.	be	827
19.	can	789
20.	it	769
21.	data	761
22.	from	755
23.	drives	751
24.	or	750
25.	an	712

Figure 3.7 The top 25 words by frequency in the optical storage media corpus

Only when we reach the 14th and 17th words (*CD-ROM* and *drive*) do we begin to see anything really useful about the terms in this text. Once the stop words are removed we get the following list of *possible* terms:

```
 1. CD-ROM          1006
 2. drive           849
 3. data            761
 4. drives          751
 5. storage         639
 6. CD              596
 7. software        584
 8. disk            579
 9. optical         475
10. system          395
11. Title           350
12. Inc             333
13. users           319
14. time            313
15. disks           299
16. access          279
17. files           273
18. available       251
19. file            250
20. performance     248
21. systems         245
22. video           238
23. information     236
24. Windows         234
25. multimedia      231
```

Figure 3.8 The top 25 words by frequency with the stop words removed

The high rank of *Title* is puzzling until we examine the text and find that it occurs at the beginning of every article. This shows a problem of having a corpus that is too small or that comes from too few sources: a particular quirk of one text has biased the statistics for word frequency.

The technique of word frequencies allows us to find terms that are single words but, as Aussenac-Gilles et al. point out, most terms are compound words – they are made up of more than one word. [Aussenac-Gilles et al.]. The list of compound nouns found as a preprocessing step (see section 3.3.1.3) gives us one list of possible multi-word terms. Using the frequency operations can find additional terms that don't consist of just nouns.

The frequency operations that find word pairs and word triples find all pairs (bigrams) and triples (trigrams) of adjacent words in the text. (Notice that there is no particular part of speech required as in the compound noun finding operation). The user can input a cutoff value as the minimum frequency desired as for the single word frequency operation. The output is ordered either alphabetically or by frequency.

The following list shows the top 20 word pairs by frequency from the same file about optical storage media.

```
 1. CD ROM                  173
 2. CD-ROM drives           145
 3. CD-ROM drive            114
 4. optical disk             95
 5. hard disk                78
 6. optical storage          87
 7. Photo CD                 60
 8. disk drive               58
 9. transfer rate            53
10. data transfer            52
11. storage management       52
12. access time              51
13. disk drives              50
14. Pinnacle Micro           47
15. hard drive               43
16. disk space               40
```

```
17. List Price                  39
18. rewritable optical          39
19. optical drive               38
20. optical drives              38
```

Figure 3.9 The top 20 word pairs by frequency

The list contains some interesting phrases that should be investigated such as *optical disk, access time,* and *optical storage.* If the user is familiar with the domain it will be fairly obvious which of these suggested terms are actually valid terms. If the domain is a new one for the user, each of these proposed terms may have to be looked at in more detail (by examining sentences in which they occur) before eliminating or deciding to keep them as terms.

Now let's look at the word triples from the same file:

```
 1. data transfer rate                26
 2. CD ROM drives                     26
 3. average access time               23
 4. hierarchical storage management   20
 5. CD ROM drive                      19
 6. optical disk drive                18
 7. Pinnacle Micro Inc                18
 8. Wall Street Journal               17
 9. Warner Interactive Group          15
10. rewritable optical disk           15
11. Time Warner Interactive           15
12. Storage Systems Division          14
13. Optical Data Library              14
14. optical disk drives               14
15. IBM Storage Systems               13
16. Eastman Kodak Company             12
```

```
17. average seek time                12
18. Phillips Consumer Electronics    12
19. digital audio tape               12
20. multisession Photo CD            11
```

Figure 3.10 The top 20 word triples by frequency

This list contains many interesting terms such as *hierarchical storage management, data transfer rate* and *rewritable optical disk* plus quite a few company names like *Pinnacle Micros Inc* and *Eastman Kodak Company.*

Note that the frequencies of the word pairs and word triples are less than those of the single words. It must be kept in mind that the more frequent a possible term, the more likely it is to be correct. When a possible term only occurs 5 or 10 times in a large text, there is the possibility that it is being used incorrectly, or only by one particular author. If the text itself is not very large a low frequency term is again not very reliable. The more *evidence* of the use of a term we can find, the more confident we are of its correctness.

If the input text has been tagged by part of speech, the one word frequency operation can also generate lists of words limited by part of speech, only nouns, only verbs or only adjectives. The following list shows some of the verbs from the same text, this time sorted alphabetically:

```
access       40
accessed     31
add          49
added        42
adds         25
allow        31
allows       57
announced    51
automated    23
```

```
based                90
begin                31
bring                23
```

Figure 3.11 A portion of the verb frequency table

Since the list is arranged alphabetically, we see that different forms of the same verb are listed separately: *add, added* and *adds.* A valuable addition to the operation would be to combine the various inflected forms of the verb.

It is also interesting to note that even the most frequent verbs occur much less frequently than the most frequent nouns in the same text. The five most frequent verbs are:

```
1. offers    129
2. include   126
3. includes  119
4. get       118
5. provides  117
```

Figure 3.12 The five most frequent verbs

whereas the five most frequent nouns are the top 5 words from the one word frequency count that we saw before are:

```
1. CD-ROM          1006
2. drive           849
3. data            761
4. drives          751
5. storage         639
```

Figure 3.13 The five most common words of any part of speech

It should be clear that the analysis of the frequency of words in a text tells us a lot about the contents of the text, as well as a lot about the structure of language.

3.3.4 Concordance

Once one or more of the terms have been found in a document, the next step of the analysis can be done: a more in-depth examination of the concepts in the text named by the terms. A lexicographer might, at this stage, want to see how the terms are used in sentences; someone who was trying to understand the ideas in a document would want to find out what the terms mean; someone who is verifying existing documents would want to check that the terms are used consistently and correctly.

The most general tool used for doing all these things is the *concordancer*. A *concordance* is a list of all occurrences of a word (or pattern) in a text, displayed with a part of the surrounding context. This allows the user to quickly see all the ways a particular word is used, instead of searching through the text manually or using a text editor.

Two kinds of concordance are provided as part of the Text Analyzer: KWIC and sentence. A KWIC concordance (short for Key Word In Context) limits the size of the context to a single line of the display and lines up all the occurrences of the word in one column. In the Text Analyzer, the user can choose the width of the KWIC display and the position of the column of words. A sentence concordance shows the word and the entire sentence in which it occurs. Figure 3.14 shows the first 20 lines of the KWIC concordance on the word *tape* from the optical storage media text. The actual output contains about 230 lines.

```
 177. C 's robotic autochanger    tape  backup system ) ( 1994 Pr
 178. LS ) robotic autochanger    tape  and optical backup units
 184.        Access times with    tape  drives are in the range o
 185. om $10,900 for a 4mm VLS    tape  changer with a single dri
 186. u feel more like a human    tape  swapper than a network ma
 188.  Just pop your choice of    tape  or optical media into the
 189. LS stores the individual    tape  cassettes or optical disk
 786. ny 's family of backpack    tape  drives .
1299.                  Magnetic  tape  or another form of offlin
1305. data can be offloaded to   tape  .
1311. chival media as magnetic   tape  .
1327. mply back up all data to   tape  on a daily basis .
```

```
1329. k drives , four MTI 8ram    tape  drives , and a 192-MB Sys
1330. g up each evening to 8mm    tape  .
1333.  manually located on the    tape  , a fairly labor-intensiv
1343. 1 TB of data from an 8mm    tape  to magnetic disk .
1389. ated movement of data to    tape  , automated operations an
1417. isk and offline magnetic    tape  lies a central component
1549. a series of OEM disk and    tape  drives , as well as an ex
1569.                             Tape  Measure Digital also anno
```

Figure 3.14 KWIC concordance on the word *tape*

To further aid the user in quickly extracting information from a concordance, the lines or sentences can be sorted alphabetically by the previous or following word. Figure 3.15 shows a section of the KWIC concordance on *tape* sorted by previous word and figure 3.16 shows it sorted by following word.

```
6237. ired by Shell , says DAT    tape  was too slow and too manu
2559.  drive , a 140G-byte DLT    tape  auto-loader , a 1G-byte d
2562.  and 140G-byte TZ877 DLT    tape  auto-loader support Novel
9252. ily of automated desktop    tape  changer .
7423. , a dropout on a digital    tape  is an abstraction , a pai
2556.  140Gbyte digital-linear    tape  drives , the latter inclu
1589. isk , solid-state disk ,    tape  and optical devices compa
9347.  migrate files from disk    tape  and back again , create a
9291. y appear higher than for    tape  systems .
9344.     Implementing HSM for    tape  backup and recovery invol
9281.        The company found    tape  storage too inconvenient
6827.  data , with two to four    tape  drives and as many as 100
6830. spread among two to four    tape  drives will cost between
6382. kes by staging data from    tape  to disk .
2486. nt manager for half-inch    tape  cartridges at St. Paul ,
9330. ges for 8mm helical-scan    tape  and 4mm digital audio tap
9337. ges for 8mm helical-scan    tape  and 4mm digital audio tap
9348. jukebox or high-capacity    tape  carousel you 've chosen .
 186. u feel more like a human    tape  swapper than a network ma
6848.  been on improvements in    tape  technology , optical stor
```

Figure 3.15 KWIC concordance on *tape* sorted by the previous word

```
9321.  comparable to that of a    tape   backup system .
9333.                             Tape   backup software packages
9344.     Implementing HSM for    tape   backup and recovery invol
5872.  ago about the demise of    tape   , but that 's not what 's
4291. neither floppy disks nor    tape   can duplicate .
9313.  said that in some cases    tape   can have a four-to-one co
9348. jukebox or high-capacity    tape   carousel you 've chosen .
9345. placed with jukeboxes or    tape   carousels ; sequential st
2473. nd marketing of the 3480    tape   cartridge to 3M ; optical
9247. s of data on a 120-meter    tape   cartridge when using the
2475.  Once Read Many ( WORM )    tape   cartridges used in the IB
2486. nt manager for half-inch    tape   cartridges at St . Paul ,
2567. which supports up to 264    tape   cartridges .
9248. th 60-meter and 90-meter    tape   cartridges .
 189. LS stores the individual    tape   cassettes or optical disk
 185. om $10,900 for a 4mm VLS    tape   changer with a single dri
9252. ily of automated desktop    tape   changes .
9440. e of robotic devices - -    tape   changes , optical jukebox
6823.              Automating     tape   changes is relatively new
7356.                  Better     tape   coatings , typically usin
```

Figure 3.16 KWIC concordance on *tape* sorted by the following word

Sorting a concordance is helpful in several ways. Sorting on the previous word can help us discover what *kinds* of tapes there are (*DAT tape, 8mm tape, audio tape, backup tape, 8mm helical-scan tape, magnetic tape* etc.). Since each of these appears together because of the alphabetical sorting, it is easy to scan the concordance and see which ones appear more frequently, possible indicating their relative importance. By sorting on the following word we can find the things associated with tapes (*tape jukebox, tape library, tape recorder, tape swapper, tape cartridge, tape drive* etc.). For other kinds of concepts we can also find out the activities or functions of the concept (*dog eats, dog sleeps, dog chases cats* etc.) or the parts of the concept (*bicycle wheel, bicycle seat, bicycle chain* etc.).

If the input file is tagged with parts of speech, the user can choose to show the parts of speech in the concordance display, or to remove the tags and just show the words.

Another version of the concordance operation searches for sentences containing two separate search words, for example: What are all the sentences containing both *doctor* and *hospital* ? The distance between the two words can be restricted if desired and the order of the words can be specified. This operation is useful for finding the relationship between two concepts. The two word concordance is available as a KWIC or sentence concordance.

A sentence concordance operation can also be used as a filter operation to pick out of the input file only those sentences containing the word of interest. This can be saved as a file and used as input to further operations, reducing the processing time because the new input file is smaller than the original.

3.3.5 Compounds including the search word

Another way to analyze a particular word is to use the operation that finds compounds including that word. This operation finds all continuous sequences of nouns and or adjectives containing the search word. For the word *disk*, we find the following list of compounds and their frequencies:

```
 1. disk space                              18
 2. disk drives                             11
 3. optical disk drives                      7
 4. hard disk drives                         5
 5. floppy disk drive                        4
 6. optical disk cartridges                  4
 7. disk grooming                            4
 8. OD152 rewritable optical disk drives     3
 9. hard disk drive                          3
10. disk storage                             3
11. disk capacity                            3
12. disk subsystem                           3
13. disk drive                               3
14. magneto-optical disk drive               3
15. free disk space                          3
16. optical disk system                      3
17. disk caddy                               3
```

```
18. disk formatting                          2
19. available hard disk space                2
20. magnetic disk storage                    2
21. Winchester disk drives                   2
22. disk diameter                            2
23. hard disk servers                        2
24. floppy disk version                      2
25. rewritable optical disk market           2
26. AMASS optical disk jukebox file system   2
27. disk speeds                              2
28. optical disk subsystem                   2
29. disk use                                 2
30. large-capacity hard disk drive           2
```

Figure 3.17 Possible compounds including the word *disk*

3.3.6 Collocations

The One Word Neighbourhood operation is based on ideas from Xtract [Smadja, 1993], described in section 2.2.4. Given a word, it finds all the other words in the neighbourhood of the search word (within 5 words on either side) and their frequencies, and displays the output as a table. If two words are a collocation they will appear together frequently and they will appear in a rigid form such as one always following the other. Because of this we can spot collocations in the output table by looking for high frequency occurrences.

Figure 3.18 shows the output for the word *disk* showing collocates that appeared 10 or more times. The output is divided into three tables: one each for nouns, verbs and adjectives. The numbers from -5 to 5 at the top refer to the position that the word appeared at in relation to the search word *disk*. For example, the word *drives* appeared 65 times as the word following *disk* as in *disk drives*.

NOUNS	TOTAL FREQ.	-5	-4	-3	-2	-1	1	2	3	4	5
drives	76	3	0	2	1	0	65	2	2	1	0
drive	74	0	5	2	0	0	63	0	0	1	3
space	44	0	1	0	1	0	40	0	1	1	0
storage	39	2	4	1	1	0	16	0	6	8	1
data	26	5	8	4	1	0	0	0	2	3	3
tape	23	2	3	1	1	0	1	5	4	1	5
disk	22	3	3	4	1	0	0	1	4	3	3
you	21	2	2	3	0	0	0	4	5	2	3
Magneto-optical	21	0	0	0	0	21	0	0	0	0	0
files	21	6	4	1	4	0	1	0	0	3	2
CD-ROM	20	1	3	0	0	12	0	0	0	3	1
it	19	3	4	1	0	0	0	2	2	3	4
capacity	18	0	3	1	1	1	4	1	1	4	2
Keywords	15	4	0	11	0	0	0	0	0	0	0
file	14	1	1	3	1	0	2	2	0	4	0
"	14	1	1	3	0	1	0	2	4	1	1
system	13	0	2	1	0	1	3	1	2	3	0
Product	12	1	0	4	0	0	0	7	0	0	0
market	11	0	2	0	0	0	7	2	0	0	0
server	11	1	0	5	2	1	2	0	0	0	0

VERBS	TOTAL FREQ.	-5	-4	-3	-2	-1	1	2	3	4	5
is	67	9	4	5	2	0	16	17	7	4	3
be	28	4	5	1	0	0	0	8	5	2	3
are	17	4	4	0	0	0	3	3	2	1	0
has	12	0	0	0	2	1	2	2	0	2	3

ADJECTIVES	TOTAL FREQ.	-5	-4	-3	-2	-1	1	2	3	4	5
optical	110	2	1	0	1	98	0	3	1	3	1
hard	81	0	1	0	0	78	1	0	0	0	1
magnetic	24	0	1	0	0	20	0	1	1	0	1
single	21	1	0	4	3	13	0	0	0	0	0
floppy	20	0	0	1	0	17	0	1	0	0	1
rewritable	18	1	0	0	15	0	0	0	0	1	1
available	16	0	1	3	2	1	0	4	3	2	0
magneto-optical	14	0	1	0	0	13	0	0	0	0	0
3.5-inch	14	1	3	4	5	1	0	0	0	0	0

Figure 3.18 Output of the one word neighbourhood operation for the word *disk*

To spot a collocation in this output we must look for a word that appears a large number of times in one position and very few in the other positions. In the nouns table, *Magneto-optical* fits these requirements perfectly – it occurs 21 times with *disk* and only appears in the -1 position as in *Magneto-optical disk*. We can pretty confidently say that this is a collocation. Other probable collocations from this data are: *disk drives, disk drive, disk space, disk storage, CD-ROM disk, optical disk, hard disk, magnetic disk* etc. These have most of their occurrences appearing in one position with a few in other positions.

Pairs of words that are related but are not real collocations appear frequently together but not with a marked frequency increase in one relative position and a decrease in the other positions. The word *file* is not a collocation because it appears spread out among the various word positions fairly equally, although it may be related. We can imagine using *file* and *disk* together in sentences such as:

1. That is a disk file. (*file* in position +1)
2. Do you have this disk on file? (+2)
3. Do you have this file on disk? (-2)
4. My disk contains a file that is very large. (+3)
5. There is only one file on this disk. (-3)

The quotation mark " which has sometimes been mistakenly tagged as a noun, also appears spread out over the positions indicating that it is not a collocation, which we know is true.

The verb table does not gives us much interesting information because all the verbs are variations of *to be* or *to have* which do not carry much meaning by themselves. An improvement to this operation would be to give the user the option to reject any words included in the list of stop words used by the frequency operation.

3.3.7 Intervening word

The Intervening Word operation works in conjunction with the One Word Neighbourhood operation to find what words occur between two search words. For example, in the output for the One Word Neighbourhood operation in figure 3.18 for the word *disk*, the word *rewritable* occurred 15 times in the -2 position. We would like to know what word occurred in the -1 position (*rewritable ??? disk*). The user enters the two search words (in order) and the maximum distance in words between the two search words. If the input text is tagged, the user can restrict the part of speech of each search word and the words in between if desired. In the example of *rewritable* and *disk* we found that the intervening word was always *optical* as in *rewritable optical disk* . This is a three word collocation.

3.3.8 Verbs following the search word

Another way to find out more about a concept denoted by a noun is to look at the verbs that follow it and the adjectives that precede it.

The verbs that have the noun of interest as their subject may identify properties (or characteristics) of the concept that the noun represents. In addition, the objects of the verbs are the value of these properties. Looking at a small text about hydras (tiny invertebrate animals) we find such phrases as:

```
1.     The hydra lives in          ponds, lakes, and streams.
2.        Hydras react to          a variety of stimuli...
3.  ...the hydra does not chase    its prey...
4.       A hydra can eat           animals which are very large...
5.        Hydras reproduce         asexually by budding.
```

from chapter 7 of *Animals Without Backbones* by Ralph Buchsbaum

Given a noun, the Text Analyzer can present the following verbs in two ways: as a list of the verbs with their frequencies and as the actual sentences, sorted by the following verbs. The frequency information is important because the more often a particular verb appears with the word, the more evidence that this verb is important and correct. A verb that appears only once with a noun in a large text may be used in a humorous or sarcastic way ("A hydra would *watch* TV if it had eyes") or it may be simply incorrect.

The following list shows all the verbs (with their frequencies) that occurred more than three times after the pattern *drive** (the star indicates all words that start with *drive*) in the optical storage media text:

```
 1. offers        12
 2. based          7
 3. supports       7
 4. use            6
 5. offer          6
 6. provides       6
 7. read           5
 8. play           5
 9. 's             5
10. comes          5
11. reads          4
12. using          4
13. were           4
14. includes       4
```

Figure 3.19 Verbs that occurred after the pattern drive*

The appearance of *'s* may seem strange but it is a verb as in *He's gone to the park* where it is short for *has.* The tagger separates the *'s* from the word it belongs with so it appears as a separate word.

This operation also outputs the sentence in which these verbs occur. Here are a few of the sentences found for the verb *read.* The sentence number is in brackets after the sentence.

```
-------------------------------------------
read

* The drive can still read and write data in the 256-megabyte
and 128-megabyte format used on MOST 's earlier products.
(3583.)

* Personal Scribe handles all formatting necessary to create
an ISO 9669-standard CD that any CD-ROM drive can read .
(3701.)

* To determine a drive's average data-transfer rate, we
commanded the drives to read data clusters of varying sizes,
ranging from one to 200 2K sectors. (7016.)
-------------------------------------------
```

Figure 3.20 Sentences for the verb *read* after the pattern *drive**

The Text Analyzer picks out verbs that are within a certain number of words (e.g. 3) after the noun because there may be modifying words between them. It is as important to know that *the drive **cannot** read CD-ROM's* as it is to know that *the drive **can** read CD-ROM's.* The sentence is rejected, however, if there is a punctuation mark between the search word and the verb (*The dog, whose mother barked a lot, was brown* – the dog we are interested in is not the one that barked a lot).

Two verbs, *to be* and *to have* are treated as special cases and are not included in the general verb following noun operation because of the large numbers of sentences that contain these two verbs. There are separate operations each for the verbs *to be* and *to have.*

To be: Forms of the verb *to be* may indicate an *isa* relationship but may also be used to describe properties as do other verbs. Looking at the same text as before and using the noun *hydra*, we find:

```
1. ...the hydra is    still a relatively small animal.
2.       Hydras are   abundant...
3.    If a hydra is   touched, the sensory cells...
4.     The hydra is   also called a 'polyp'...
```

from chapter 7 of *Animals Without Backbones* by Ralph Buchsbaum

Sentence number 1 gives a true *isa* relationship (a hydra is an animal) but the other sentences do not. Sentence 2 gives a property, sentence 3 shows what happens to a hydra when something is done to it, and sentence 4 provides a synonym of the word *hydra*, something that is also very important to establish.

When run on the optical storage media text with the search pattern *drive**, this operation finds 192 occurrences of the verb *to be*. A few of the sentences found are:

```
-----------------------------------------------
are

* Magneto-optical (MO) drives are a relatively affordable,
safe secondary storage option that can serve as main drives
in a pinch. (4281.)

* Although optical drives are fairly costly, they can be
useful for fast information retrieval. (6807.)

* Triple-speed drives are desired by serious multimedia and
game fans. (1051.)
-----------------------------------------------
is

* Using several drives at once is also thought to speed up
the backup process. (6845.)
```

```
* Maxoptix's Tahiti IIM 1GB optical disk drive is now
compatible with Microsoft Corp.'s Windows NT operating
system. (6471.)

* Installation is very easy, and the drive is extremely
quiet. (5314.)
-------------------------------------------
```

Figure 3.21 Forms of the verb *to be* occurring after the pattern *drive**

To Have: The verb *to have* indicates many different kinds of relations between its noun and its object. For instance, it marks attributes of various sorts (*She has a pleasant personality.*) and related things (*He has a daughter).* It often yields examples of the parts of the object in question. In our hydra example are the sentences:

```
1.  ...the many-celled hydra has  a network of nerve cells...
2.                      Hydra had  nine heads; and when Hercules
                                   cut one off, two grew in its
                                   place
```

from ch. 7 of *Animals Without Backbones* by Ralph Buchsbaum

The first sentence does indicate a part of the hydra: a network of nerve cells. Sentence 2 refers to parts of the mythical creature that the hydra was named after. In a large text about invertebrate animals, indications that a hydra has nine heads would probably appear only once or twice whereas sentences containing references to the network of nerve cells would probably be frequent, evidence that this kind of hydra has a network of nerve cells and not nine heads. This shows the need for human judgment in the process of extracting knowledge using the Text Analyzer.

These operations miss a good deal of information stored in the text. One of the main problems is that a pronoun or hypernym may replace the search word in an important sentence as in the following example:

1. The simplest method of locomotion in a hydra is a gliding on the base due to a creeping amoeboid movement of the basal cells.
2. *The animal* bends over.
3. *It* places its tentacles on the ground.

In sentence 2 *hydra* has been replaced by the hypernym *The animal* and in sentence 3 it has been replaced with the pronoun *It*. In other sentences no direct reference is made to the search word at all:

1. Digestion takes place in the interior cavity.

This is a problem not yet addressed by the Text Analyzer.

This operation finds 27 occurrences of the verb *to have* on the optical storage media text after the word *drive**. Some of the sentences found are:

```
-----------------------------------------------
has

* Sony 's $6,500 CDW-900E is a double-speed CD-R drive that
can be placed in a daisy chain of up to 17 drives and has a
4Mbyte data buffer.(3954.)

* Now shipping, the drive has a list price of $4995. (4860.)

-----------------------------------------------
have

* We found that some cartridges formatted by older drives
needed to have Microtech 's driver installed on them to mount
properly. (5339.)
```

```
* The drives also have automatic lens-cleaning features and
dust-protection doors. (8453.)

-----------------------------------------------
```

Figure 3.22 Forms of the verb *to have* occurring after the pattern *drive**

3.3.9 Adjectives preceding the search word

The adjectives preceding the search word operation is similar to the verbs following the search word operation. This finds all adjectives that immediately precede the search word. This is useful for identifying subconcepts and properties of the search word.

For the search word *drive**, we find these adjectives occurred for than 10 times:

```
 1. optical              111
 2. hard                 93
 3. double-speed         57
 4. magneto-optical      43
 5. new                  37
 6. other                31
 7. rewritable           27
 8. 3.5-inch             27
 9. external             24
10. 5.25-inch            21
11. removable            17
12. internal             16
13. floppy               13
14. quadruple-speed      13
15. triple-speed         13
16. single               12
17. Floptical            11
18. first                11
```

Figure 3.23 Adjectives occurring before the pattern *drive**

3.3.10 Conceptual operations

All the preceding operations aim to find out how the *words* in the text behave. From their output the user can deduce the meanings of and relationships between concepts. The conceptual operations aim to directly find this kind of information from the text by means of clues in the structure of the sentences. For example, the phrase *is a kind of* as in *The rose is a kind of flower* indicates the relation of *hyponymy* or inclusion of one class in another.

Ahmad and Fulford explored this idea in [Ahmad and Fulford, 1992] with the aim of extracting semantic relations from text for the purpose of building term-banks. They refer to the clues as *knowledge probes*. Ahmad and Fulford concentrated on finding knowledge probes indicating the relations of synonymy, hyponymy, part-whole, cause and effect, and material (the relationship showing what something is made of).

Marti Hearst concentrated on looking for hyponyms in [Hearst, 1992]. Hearst wanted to avoid the need for pre-encoded knowledge and to find a method that is applicable across a wide range of texts.

There are four conceptual operations implemented in the Text Analyzer. The first attempts to find super and subconcepts of the concept (hypernyms and hyponyms) the second looks for synonyms or explanations of a concept, the third tries to locate the parts of the concept and the fourth searches for the functions of the concept.

The **find possible super and sub concepts** operation first finds all sentences in which the search word occurs close to (within 25 characters) one of the key phrases shown below with examples:

1. *is a, is an, are a, are an*	Windows NT *is a* 32-bit operating system.
2. *such as*	Operating systems *such as* Windows NT ...
3. *and other*	Windows NT *and other* 32-bit operating systems...

4. *including*	Operating systems, *including* Windows NT...
5. *especially*	Most operating systems, especially Windows NT...

These phrases are the ones found by Marti Hearst [Hearst, 1992]. All these examples show a super-sub relationship between *Windows NT* and *operating system* but these patterns can also turn up incorrect sentences.

Further checking must be done to eliminate some sentences where the relationship is not that of super-sub. If the pattern found is "... preposition (determiner)*word* ISA..." , where ISA stands for any of the key phrases, the sentence is discarded to eliminate phrases such as *The dog in the car is a German Shepherd.* If the search word is *car*, the phrase *car is a German Shepherd* is picked out but obviously this makes no sense – it is the dog that is the German Shepherd. If, however, the search word had been *dog*, the sentence would be kept.

A second check is made to look for the pattern "...*word* ... , ... ISA ...". Sentences with this pattern are also discarded unless the pattern is "...*word,* which/that/who/ ISA...". This eliminates sentences such as *The big dog, whose owner is a woman, barked at me.*"" but keeps sentences such as "*The big dog, which is a German Shepherd, barked at me.*"

The results of this operation still must be examined by a human being since many of the suggested supers are not correct. Some appear because the key phrase has been used in a different way than super-sub. The accuracy of these may be improved with more filtering in the operation but other sentences are picked because they meet all the criteria but don't quite say what we need.

Figure 3.24 shows some of the 39 lines returned by this operation with the search word *CD-ROM* in the optical storage media corpus:

```
6044.   as CD-Interactive and   CD-ROM  XA which are incompatible and lend
6133.         Superstore is a   CD-ROM  disk that contains encrypted softw
7217.                           CD-ROM  is an ideal delivery medium for th
7258.  e paper version , this   CD-ROM  is an excellent alternative .
```

```
7267.   : $49 Flight 642 is a   CD-ROM  with literally thousands of files
7306. lopedia is a must-have    CD-ROM  .
7580. SI peripherals such as    CD-ROM  drives to a PC is a catch-as-catch
7890.  Spring is a brilliant    CD-ROM  .
8029. y House is an engaging    CD-ROM  for the under-five set that lets k
8212. , including CD-Audio ,    CD-ROM  , CD-ROM-XA , and CD-I formats .
8391. s heart , this Windows    CD-ROM  is a storyteller , and an excellen
```

Figure 3.24 Possible super and subconcepts for *CD-ROM*

In this text, since it is assumed that the readers already know what CD-ROM is, most of these sentences indicate subconcepts of CD-ROM, such as Superstore in sentence 6133. or Flight 642 in sentence 7267. As far as finding out what CD-ROM is, the sentences returned by this operation tell us that it is a device, a publishing medium, a delivery medium for information, and an excellent alternative to paper.

The **find synonyms** operation is similar to the supers and subs operation and searches for the key phrases shown below with examples:

1. *called, Called* The German Shepherd, *called* the Alsation in England, ...

2. *known as, Known as*

This large, white, scented flower, *known as* the moonflower, ...

3. *referred to as, Referred to as*

...the ServicePak, *referred to as* Corrective Service Diskettes in IBMspeak, ...

It finds both true synonyms, such as in examples 1 and 3, and descriptive explanations, such as in example 2.

In the optical storage media corpus we searched for explanations of MultiSpin 3Xp. The results are show in figure 3.25.

```
105.  table version of the CD-ROM drive , called the  MultiSpin 3XP  .
382.  on , a digital DiscMan-impersonator called the  MultiSpin 3XP  .
```

Figure 3.25 Possible synonyms or explanations of *MultiSpin 3XP*

The **find parts** operation attempts to find the parts that make up the concept in question. For example, a chair has the parts *seat, back* and *legs*. The operation searches for the following phrases close to the word:

1. *part, parts* The kernel is *part* of the operating system.

2. *designed with* The car is *designed with* an aerodynamic body.

3. *contain** The fruit *contains* small, black seeds.

4. *includ** The mechanism *includes* three springs and a motor.

and

5. possessive Her *dog's* nose was cold and wet.

For the pattern *drive**, this operation returned 39 sentences. A section of the output is shown in figure 3.26

```
 951.                    The   drive   includes an automatic lens-cleanin
1099.  been including CD-ROM   drives  in their systems , but PC makers s
1965.  two-piece gizmo : the   drive   snaps into a base containing a bat
2327.           The included   drivers cover most popular SCSI-2 adapters
2333. ging large numbers of    drives  exist -- including Meridian Data's
2686. ackage includes a CD-R   drive   manufactured by Ricoh, a power cor
3241. e bundled with several   drives  and CPUs , including the Apple Per
4467. ning to include CD-ROM   drives  , magneto-optical 's emergence in
4501.                   The    drive   does not include cables , a SCSI i
```

```
5120. de double speed CD-ROM  drives  ) (Reveal Computer Products ' MFX0
5127.                   Each  drive   includes a 16-bit StereoFX Studio
5332. y Mac with a removable  drive   bay , including the Mac IIvx and a
```

Figure 3.26 Possible parts of *drive**

The **find functionality** operation tries to find what the thing that the search word represents does. The key phrases here are:

1. *provide** The software package *provides* spell checking.

2. *offer** The Korg keyboard *offers* automatic key transposition.

3. *support** Some operating systems *support* multitasking.

4. *uses, using, used by* Windows NT *uses* a flat memory model.

5. *includ** Features *include* rapid acceleration and easy navigation.

It can be seen that both the parts and functionality operations look for the phrase *includ** because this can indicate both relationships. It is up to the user to decide whether the sentences found by the operation are correct for each case.

This operation returned 54 sentences for *drive* in the optical storage media corpus. A section of the output is shown in figure 3.27.

```
  59.  CD-ROM  drive offers a triple-speed mechanism with an average acce
 614. e-speed  drive offers an average access time of less than 180 milli
 615. d , the  drive provides about one hour of audio CD playing time or
 618.     The  drive supports multisession Photo CD and includes a 64-Kby
 641.     The  drive supports discs in Macintosh and IBM PC and compatibl
 781.  CD-ROM  drive that supports data transfer rates of 300KB per secon
 830.     The  drive offers a 306K-bps data-transfer rate and an access s
 947.  CD-ROM  drive , which provides data transfer rates of 600 Kbytes p
 951.     The  drive includes an automatic lens-cleaning system , dual SC
 983. -change  drive providing an average seek time of 45 milliseconds an
```

```
 991.  single  drive to access its 28 cartridges .
1256. CD-ROM   drive provides much smoother playback of animations and vi
```

Figure 3.27 Possible functionality of *drive**

3.3.11 Utilities

Several utilities are available to make the process of knowledge extraction from the document easier.

3.3.11.1 File operations

Three file management operations allow the user to create, view and remove files from within the Text Analyzer.

The **save file** menu option allows the user to save the output of an operation as a file for later viewing or reuse as input to another operation. The text may be edited before saving with copy, cut and paste or by typing additional notes or text. The Text Analyzer automatically suggests a name for the file based on the operation performed and the search word used. The suggested names all have the suffix *.ta*.

The **view file** menu option brings up a list of files saved previously and lets the user pick one for viewing in the lower window of the Text Analyzer. Only files with the *.ta* suffix are shown in the list.

The **delete file** menu option allows the user to delete files from the list of *.ta* files when they are no longer needed.

3.3.11.2 Hardcopy

The hardcopy operation causes the text in the lower window to be printed out. The user is prompted for the name of the printer to use. The text can be edited before printing if desired.

3.3.11.3 View larger context

This operation lets the user view the text surrounding a particular sentence. The number of sentences before and after the main sentence can be set by the user. The user is asked for the sentence number (or the number can be selected from the text in the lower window) and the larger context of the sentence is then displayed in a separate window.

This operation is useful because in a concordance, for example, the best information may not be in the actual sentence containing the search word. Take the following sentences:

```
13. Let's look at the Siberian Husky.

14. This medium-sized dog is often used for dog sledding and
is renowned for its endurance and strength.
```

If a concordance is done on text containing these two sentences with the search word being *Siberian Husky*, only sentence 13. would be returned because sentence 14 does not contain the search word. All the useful information in sentence 14. would be lost. If, as in this example, the sentence appearing in the concordance seems to indicate that further information follows in the next sentences, it is worthwhile to look at the larger context of the sentence.

3.3.11.4 Change input file

The user can change the corpus file being examined by typing in the name of the new file. However, the user also has the option of having several Text Analyzers open at the same time on different files.

3.4 The link to CODE4 – Building a knowledge base

The user has the option of building up a CODE knowledge base from the knowledge found in the text. The resulting knowledge base can be used as a reference, as a summary of the knowledge in the text or as a teaching aid to introduce the subject to others.

If the user chooses to make a knowledge base, she adds each piece of knowledge (called a statement) to the knowledge base as it is found by the Text Analyzer. Figure 3.28 shows a Text Analyzer connected to a CODE4 knowledge base. The bottom section of the Text Analyzer has a template for assembling statements consisting of an area for each of the subject, the property and the value. The subject of the statement is the concept that the statement is about: *CD-ROM, dog* etc. The property is also known as the characteristic and is what is being said about the concept: *price, colour* etc. The value is the value of the property: *$450* or *brown.* Together they make up one statement about the concept: *The price of the CD-ROM is $450, The colour of the dog is brown.* In figure 3.28 the subject is *Unix* and the property is *kernel.* There is no value because the statement says only that *Unix has a kernel .*

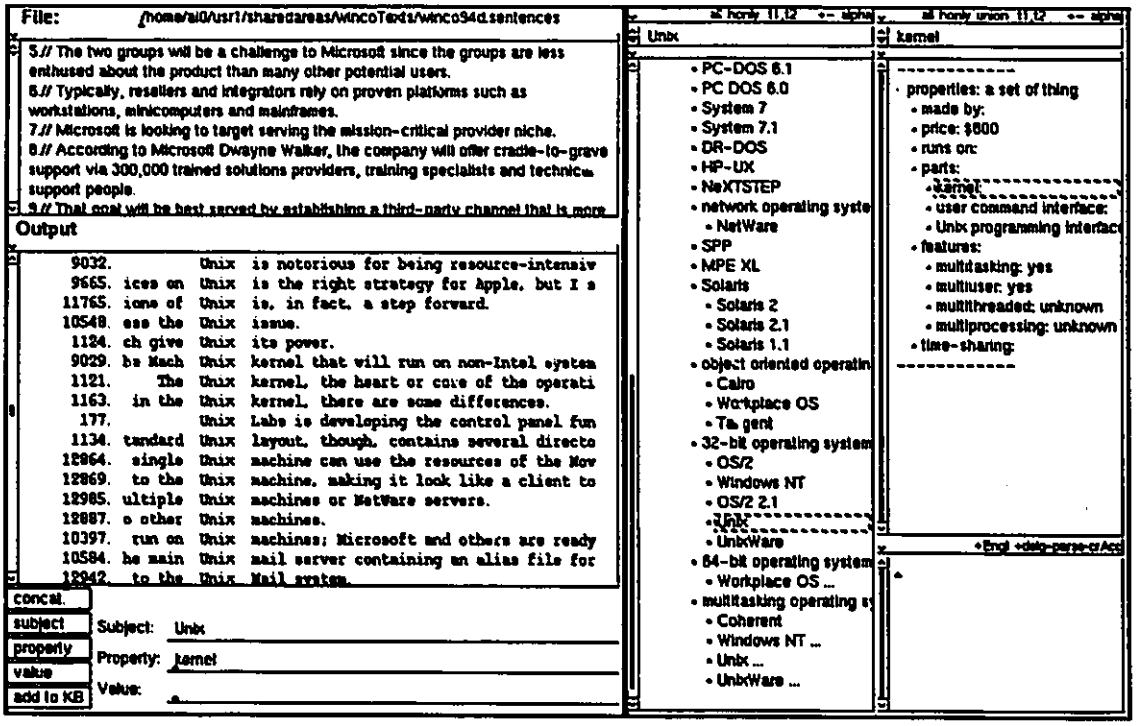

Figure 3.28 The Text Analyzer connected to a CODE4 knowledge base

To assemble a statement on the template, the user fills in the concept, property and value by one of three methods:

1. Highlight the text in the output window above and click on the **subject**, **property**, or **value** buttons to the left of the template. If the text for one line is in two different places, transfer the first piece of text, then use the **concatenate** button to append the second part onto the end of the first.

2. Copy and paste the text from anywhere in the Text Analyzer

3. Type the text directly into the template.

The user does not always have to add all three of subject, property and value to the knowledge base. It is valid to add just a subject, with no property or value, or a subject with a property and no value.

Once the text is ready, the user clicks the **add to KB** button to the left to add the statement to the knowledge base. If the statement has a property, the name of the text file is added to the knowledge reference facet of the statement in the knowledge base indicating the source of the knowledge. Users are encouraged to add to this the sentence number(s) or actual sentence(s) that the knowledge came from. This is not yet done automatically. With this information, the original source of knowledge can be quickly referred to if needed.

The way a statement is added to a knowledge base depends on the type of statement the user has assembled. It may be either a statement of an ISA relationship or a statement that is not an ISA relationship.

3.4.1 Adding an ISA relationship

If the statement consists of a subject, property and value and the property is *is* or *are*, then the statement is interpreted as an ISA relationship. For example, the subject is *rose*, the property is *is* and the value is *flower*. If the property is *are*, then the Text Analyzer attempts to change the statement to the singular by removing the final s (if present) from the subject and value. For example, *roses are flowers* is changed to *rose is flower*. This is

done because the convention in CODE4 knowledge bases is to name concepts in the singular.

Some statements where the property is *is* or *are* do NOT describe ISA relationships. *Rabbits are soft* does not mean that the concept *rabbit* is a subconcept of *soft*. In this case, the property name is not very descriptive and a better way to add this statement would be to have the subject *rabbit* with the property *soft* or *softness*. Then a value could be added describing the degree of softness, such as *great*.

If the statement is a true ISA relationship then the Text Analyzer proceeds differently depending on whether the concept (the subject) and/or superconcept (the value) are already present in the knowledge base. If neither is present, it adds the superconcept and then the concept. If the superconcept is already there but the concept is not, it adds the concept as a subconcept of the superconcept. If the concept is there but the superconcept is not, the user is given a choice to:

1. Add the superconcept as an additional parent of the concept
2. Change the parent of the concept to be the superconcept
3. Add new concepts for both the concept and the superconcept
4. Cancel the operation

If both the concept and superconcept are in the knowledge base, the options available depend on their relationships to each other, If the concept and superconcept are not related (neither is a descendant of the other), the user can:

1. Add the superconcept as an additional parent of the concept
2. Change the parent of the concept to be the superconcept
3. Add new concepts for both the concept and the superconcept
4. Add another concept with the same name as the subconcept of the superconcept
5. Cancel the operation

If the concept is a descendant of the superconcept but not a child (for example, a grandchild), the user can:

1. Change the parent of the concept to be the superconcept
2. Add new concepts for both the concept and the superconcept
3. Add another concept with the same name as the subconcept of the superconcept
4. Cancel the operation

If the concept is a child of the superconcept or if the superconcept is a descendant of the concept, the user can:

1. Add new concepts for both the concept and superconcept
2. Add another concept with the same name as a subconcept of the superconcept
3. Cancel the operation

3.4.2 Adding a statement that is not an ISA relationship

If the statement to be added to the knowledge base is not an ISA relationship then the subject is added as a concept, the property as its property and the value as its value. If the subject is not already in the knowledge base, it will be added. The user is asked for the superconcept of the subject which can either be a concept that is already in the knowledge base or one that is not. If the superconcept is not in the knowledge base, the user is prompted for the superconcept of the superconcept and so on until the name of a concept is entered that is in the knowledge base. Then the subject and the chain of superconcepts is added. Next the user is asked for the superproperty of the property (if the statement contains a property) in a similar way to the superconcept. The property and chain of superproperties (if there is one) is added. Finally, the value is added if the statement contains a value.

If the knowledge base already has a concept with the same name as the subject to be added, the user is given the choice of adding another concept with the same name, using the concept already there or cancelling the operation. If the user chooses to add another concept, the Text Analyzer proceeds in the same way as if it was adding a subject not in the knowledge base. If the user chooses to use the concept that is already there and the statement also contains a property or a property and value, the Text Analyzer checks if the concept already has a property with the same name. If not, it adds the property (prompting the user for the superproperty) and the value if any. If the subject already has a property with the same name, the user can add either another property with the same name, use the existing property or cancel. If the user chooses to use the existing property and also wants to add a value, the Text Analyzer then looks at the value of the existing property. If there is al-

ready a value present, then the user can replace the existing property, append the new value to the existing value or leave the existing value unchanged

4 Implementation

In this chapter we will look at some of the underlying details of the Text Analyzer: how it was developed and how it works.

4.1 Previous work – the Knowledge Preprocessor

The idea of the Text Analyzer partly came from previous work done starting in the summer of 1992 on a tool called the Knowledge Preprocessor that was also connected to CODE4. The Knowledge Preprocessor was a first attempt at finding knowledge contained in texts. It processed the text a sentence at a time and proposed phrases in the form of one or more sets consisting of a noun or verb and its left and right modifiers. The sentences were analyzed with the help of two dictionaries: an external dictionary (the Kimmo dictionary) and a dictionary that the user built up that is stored in the CODE4 knowledge base. Statements were added to the knowledge base in a manner similar to that of the Text Analyzer. The Knowledge Preprocessor was written entirely in Smalltalk.

The Knowledge Preprocessor was not equipped to handle a large corpus and was more suited to analyzing small texts. The user had to process one sentence at a time which was slow. Analyzing a sentence was also difficult because the user was often prompted for the parts of speech of words in the sentence, something which even relatively knowledgeable users often had trouble with.

The work on the Knowledge Preprocessor was valuable, however, in that it showed that a tool to help users analyze text for the creation of knowledge bases could be useful and it paved the way for our research on the Text Analyzer.

In the next sections, we will examine the implementation of the Text Analyzer in some detail.

4.2 Smalltalk

The user interface of the Text Analyzer is written in Smalltalk. Smalltalk was chosen because it is a language well-suited to writing user interfaces and also because the CODE4 Knowledge Management System is written in Smalltalk so connecting the Text Analyzer to it would be easier.

The main emphasis of this project was on the actual operations that the Text Analyzer performs rather than the user interface, which was kept basic and simple.

4.3 Icon programs

Most of the operations that the Text Analyzer performs are written in the Icon language which is described in [Griswold and Griswold, 1990]. Icon was developed at the University of Arizona and is in the public domain. It is a high-level programming language that emphasizes string processing and accompanying data structures such as lists and associative arrays. It is very good at string scanning and pattern-matching and therefore is a good choice for text analysis programs.

We designed all our Icon programs so that they can be run without the need for the Smalltalk user interface. Since Icon is free, this means the programs are available for anyone needing to do low-cost text analysis.

One of the problems that we experienced with Icon is its inability to communicate with other languages. It does not have sockets, so in order to run an Icon program from Smalltalk, the procedure is to copy the input text to a file, get Smalltalk to start up a separate UNIX process for the Icon program which then reads in the file, processes it, and writes the output to another file. Finally Smalltalk reads in the output file and displays the results. Although this procedure works, it is, in some cases, unacceptably slow. We therefore decided to rewrite some of the programs in C.

4.4 C programs

The decision to begin rewriting some programs in C was not an easy one. Icon has a lot of features that C lacks, such as associative arrays (dictionaries), string-scanning functions, and a basic simplicity of programming. However, the slowness of communication between Icon and the Smalltalk user-interface drove us to explore other options.

The benefits of C include sockets for interprocess communication between Smalltalk and C, and its speed of operation. It is a very powerful language that can be used to do just about anything and it is very well-known and available on many platforms.

In early 1995 we started to write C code. The gain in speed was immediately apparent: a test program was written in both Icon and C that read 14,320 sentences containing approximately 17,000 different words, counted the number of occurrences of each word in the text, sorted the result by frequency and wrote the results to a file. While the Icon program took about 8 minutes to run, the C program completed this task in only 35 seconds.

Before we could actually write any programs, we had to write code for the data structures we needed, such as linked lists and dictionaries but by the summer of 1995 we had converted the basic frequency and concordance programs into C.

4.4.1 Communication between Smalltalk and C

In order to speed up the total time for an operation run from the user interface, sockets are used to communicate between Smalltalk and C. When a new Text Analyzer is started up, a separate C process is also started. The C process creates a socket and waits for Smalltalk to connect to it. Once Smalltalk has hooked up to its end of the socket, communication can begin. The C process runs continuously, waiting for Smalltalk to send requests to perform operations. With the Icon programs, every time the user wanted to perform an operation a new Icon program was started.

When the user wants an operation done, Smalltalk sends a request, according to a special protocol, through the socket to C. The C program calls the function

performFunction that takes the message received from the socket, decodes the arguments and then calls another C function to perform the request. At present, the output is written to a file which is read in by Smalltalk but the output could be returned through the socket as well.

Instead of reading the input text from a file in C each time a function is performed, the text is stored within the C program when it is first started up. Only if the user changes the input file does the C program have to read in the input file again. When an input file is read in, it is stored in an array, one sentence per entry. This allows for fast retrieval of sentences by sentence number .

4.4.2 Methods for fast text searching

In order to make searching through the entire text as fast as possible we store information about the words in each sentence that can be quickly accessed, and we use a fast pattern-matching algorithm for string searching. These techniques are described in the following sections.

4.4.2.1 Bit vectors for initial letter information

To avoid searching sentences unnecessarily, we store a piece of information about each sentence. We create an array with one long integer (32 bits) for each sentence. Each bit (with 6 left over) represents one letter in the alphabet. If the bit is a 1, it means that there is at least one word in this sentence that begins with this letter. If the bit is 0, there is no word in the sentence beginning with this letter. When searching for all the sentences containing a particular word we can, in one bit-wise AND operation, find out if there are any words in the sentence starting with the same letter as the search word. If there are not, we quickly go on to the next sentence.

Initial tests of this idea were disappointing. They showed that there was too great a probability of a match to increase the searching speed a worthwhile amount. For example, there was about a .4 probability of finding a word starting with an *n* in a sentence. This means that if we were searching for a word beginning with an *n*, we would still have to

look at 40% of the sentences. We found, however, that by creating *two* arrays, one for the words with less than 5 letters and one for words of 5 or more letters, that the probability of a match was reduced enough to get a significant speed gain. The probability of a sentence having a word starting with *n* that was less than 5 letters shrank to .28 and for words of 5 letters or more the probability was reduced even more, to .18. As an example of the speed increase this gives, finding about 4,030 occurrences of the word *operating* in a file of 55,700 sentences took an average of 12.5 seconds searching every sentence but took only 4 seconds using the bit vectors. The percentage of sentences containing at least one word starting with o with 5 or more letters is about 27%.

Let's look at an example sentence:

```
The grey cat stretched luxuriously on the chesterfield
```

This sentence contains 5 words with less than 5 letters (*The, grey, cat, on* and *the*) and 3 words with 5 letters or more (*stretched, luxuriously* and *chesterfield*). The two bit vectors for this sentence are shown in figure 4.1 with the letters that each bit represents. The bits are separated into groups of 4 for readability only.

	zx	ywvu	tsrq	ponm	lkji	hgfe	dcba
short words (< 5 letters)	0000	0000	0000	1000	0100	0000	0100	0100
long words (>= 5 letters)	0000	0000	0000	0100	0000	1000	0000	0100

Figure 4.1 Bit vectors for *The grey cat stretched luxuriously on the chesterfield*

To find out if there are any words with less than 5 letters starting with the letter t in this sentence we do a bitwise AND operation with a bit vector containing only one 1 in the position for t with the bit vector for the short words in the sentence:

```
        0000 0000 0000 1000 0000 0000 0000 0000
AND     0000 0000 0000 1000 0010 0000 0100 0100
```

The result is:

```
0000 0000 0000 1000 0000 0000 0000 0000
```

which, since it does not equal 0, indicates that there is at least one word in the sentence starting with t.

4.4.2.2 Fast string matching

When searching for a word in the entire text, as in a concordance program, we first do a bit-wise comparison to see if the word may be in the sentence. If there is a word beginning with the same letter as the search word, we then search the sentence using the Knuth-Morris-Pratt algorithm, described in [Cormen et al., 1990] and other books on algorithms, which runs in O(n + m) time where n is the length of the text and m is the length of the search word. It precomputes in O(m) time a prefix function for the search word that contains information about how the word matches against shifts of itself.

The naive pattern matching algorithm tries to match the pattern against each shift of the text, character by character. For example, if it was trying to match *cat* in the text *The cat sat on the mat* it would try *cat* against *The*, then against *he_*, then against *e_c*, then against *_ca*, and so on. This gives an algorithm of O((n-m+1)m). The Knuth-Morris-Pratt algorithm avoids examining shifts that it knows cannot be correct. If the pattern *Dadaism* matches the text *dadafzfg...* for the first 4 characters but not the 5th, the naive algorithm would then stop comparing characters, shift the pattern over one position in the text and try again. If we know, however, that *Dada...* does not match itself when shifted over 1 position, it is pointless to test this shift. We should try shifting it 2 positions, though, because here it does match itself. There is no point testing *Dadaism* against *adafzfg...* but we should test it against *dafzfg...* since we know that at least the first two letters match.

Since most words do not match any shifts of themselves at all, we can avoid a great deal of shifting and testing. For example, if the text is *cataclysm...* and the search word is *catapult*, the search word would be compared against the text first with no shifts. *cata*

would match but the next character, a *p* would not match the *c* in the text. Since *cata* does not match itself with any shifts, the pattern would be shifted immediately by 4 so that *catapult* was compared to *clysm*... The comparisons to the text as *atsclysm*..., *taclysm* ...and *aclysm*... have all been avoided.

4.5 Remarks on indexing and parsing

In designing the Text Analyzer we consciously avoided using two techniques often regarded as necessary in this kind of work – indexing and parsing. We did this because we wanted to see how useful a tool we could build without them. Indexing and parsing add a great deal of complexity and space requirements to a program and we aimed at creating a simple, but effective tool. We will explain them, however, in the following sections.

4.5.1 Indexing

The idea of *indexing* a text comes from the field of *Information Retrieval*. As Turtle and Croft [Turtle and Croft, 1992] explain "Information Retrieval is concerned with identifying documents in a collection that best match a description of a user's information need." The difference between the goals of Information Retrieval and the Text Analyzer is that Information Retrieval usually returns whole documents while the Text Analyzer searches for individual words or sentences within one (possibly large) document.

An *index* is similar to a library's card catalogue in that it is a way to choose documents from a large collection that are relevant to the user's requirements. The important keywords or topics for each document are chosen and stored together, for example, alphabetically. If the user wants to find documents on a particular topic, the index items for that topic contain pointers to where each document about that topic is stored. This is a very simplified view of indexing but the idea is the same no matter what method is used.

The main reason for using indexing is to speed up the time needed to search for documents. If the documents were not indexed, a program would have to scan through each document to see if it was relevant. For documents stored on disk, indexing is very

important in speeding up search time. Salton and McGill explain: "Disks provide rapid access to consecutive records, but access to particular regions of the disk is slow. An index may then be used to locate the special region of the disk which contains the records of interest for a given query. These records can then be scanned sequentially." [Salton and McGill, 1983]

Full-text information retrieval is an established type of information retrieval in which an *inverted file* index consists of a record, or *inverted list,* for each term that appears in the document collection [Brown, Callan and Croft, 1994]. The inverted list stores a document identifier and weight (a measure of the significance of the document) for each document where the term appears.

Creating an index for a collection of documents is a fairly complicated process. First, document parsing assigns an identifier to each document and extracts the terms from the document, eliminating stop words (like the Text Analyzer's stop words list) that are too frequent to be worth indexing. Each term is inserted into the term dictionary and gets a term identifier. After each document is parsed a temporary *transactions* file is generated that stores each term and the location of each occurrence of that term in the document. The transactions files for all the documents are then sorted and combined, or if this is too large, kept in segments. Finally, the inverted list is built for the entire document collection by reading each transaction file [Brown, Callan and Croft, 1994]

There are some drawbacks to this kind of system. Full-text indexing requires a lot of space. Brown, Callan and Croft state that the inverted lists for a multi-gigabyte document can be as large as several millions of bytes. Adding a new document to a collection also presents problems. Because of the size of the inverted file, many information retrieval systems simply re-index the whole collection. This does not encourage the addition of new documents often and to avoid this problem new methods are being developed for incremental indexing such as that of Brown, Callan and Croft described in [Brown, Callan and Croft, 1994].

For several reasons we decided not to use indexing in the Text Analyzer. First, we wanted to avoid the extra complexity. Second, we wanted to avoid the extra storage space

needed. Finally, we found the increased size of main memory and the speed of modern computers allowed us to store the entire text in main memory and to search it fast enough so that indexing was not needed.

4.5.2 Parsing

In natural language processing, parsing is the process of assigning grammatical types to each word and phrase in a sentence. A parser finds not only nouns, verbs, adjectives and so on as does a part of speech tagger but also noun phrases, verb phrases, relative clauses and others. The output of a parser is a parse tree. An example of a parse tree from [Patten, 1992] is shown in figure 4.2. The sentence is broken down first into a noun phrase and a verb phrase. The noun phrase is broken down into an article (*The*) and a noun (*boy*). The verb phrase is broken down into an auxiliary (*had*), a verb (*taken*), and a noun phrase which is in turn broken down into an adjective (*long*) and a noun (*walks*).

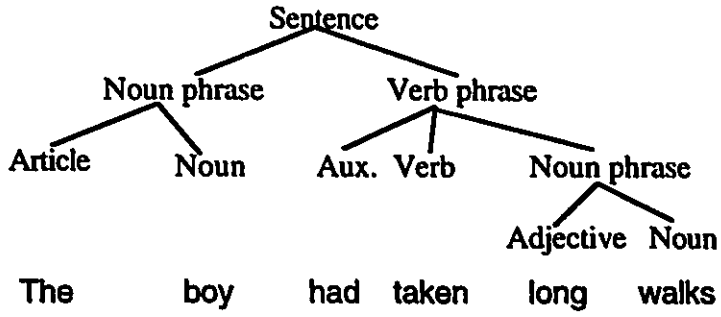

Figure 4.2 The parse tree for *The boy had taken long walks*

The same parse tree can be represented more easily by bracketing the sentence: ((The boy) (had taken (long walks))).

In order to write a parser we must have a *grammar* that explains all the rules of the language being used. In the above example, there would be a rule that states that a sentence can be broken down into a noun phrase and a verb phrase and another that says that a verb phrase can be broken down into an auxiliary verb and a verb and so on.

In computer science, parsing is part of the process that transforms computer code written in a particular language into machine instructions. This is easy to do because computer languages are very restricted in their grammar. Parsing of natural language, however, is not so easy because of the complexity of the language. As Jerry Hobbs et al. explain in [Hobbs et al., 1992], "The syntactic structure of English is very complex, and no grammar of English has been constructed that has complete coverage of the syntax one encounters in real-world texts." There are other problems with natural language parsing according to Hobbs who says: "Typical sentences in newspaper articles are about 25-30 words in length. Many sentences are much longer. Processing strategies that rely on producing a complete syntactic analysis of such sentences will be faced with a combinatorially intractable task, assuming in the first place that the sentences lie within the language described by the grammar." To make things worse, he adds: "Any grammar that successfully accounts for the range of syntactic structures encountered in real-world texts will necessarily produce many ambiguous analyses for most sentences."

Parsing, then, is a complex, time consuming process that may come up with more than one answer or may come up with none. A test performed in 1990 by Black, Lafferty and Roukos [Black, Lafferty and Roukos, 1992] rated the accuracy of four of the major large-coverage parsers for English on 35 sentences of 13 words or less from the Associated Press news wire corpus. The best one was correct on only 60% of the sentences and the others scored below 40%.

Despite these problems, many researchers, such as Hobbs et al., believe parsing is necessary for complete natural language understanding. We have taken a different approach and believe that for some tasks, it is not needed. Others agree with us.

Mikeev and Finch in their Knowledge Acquisition Workbench (described in section 2.4.2) and in [Mikeev and Finch, 1995] state that "...general and full scale parsing is not required for knowledge acquisition purposes but rather a robust identification of certain text segments is needed." On the subject of the less than perfect accuracy of parsers they say "It is worth mentioning that there is no requirements to obtain a hundred percent precision for the parser since its results will be post-processed by the knowledge engineer."

Bourigault does not use full parsing in his LEXTER system [Bourigault, 1995] for term extraction from texts, described more fully in section 2.4.1. He says "...we argued that, given the restricted grammatical forms of complex terms, it was not necessary to go into a complete syntactic analysis of the sentences to extract candidate terms from a corpus. The system uses local syntactic parsing techniques, which are based on surface patterns and satisfy the required robustness and accuracy constraints."

The message seems to be that we should not assume that parsing is necessary for all natural language tasks but we should examine the tasks we are performing and see if the same results can be obtained with simpler methods. That is what we tried with the Text Analyzer and the results so far seem to prove correct our intuition that full parsing was not a requirement. The grammatical information given by the part of speech tagger is sufficient for our purposes.

5 An example of using the Text Analyzer

In this chapter we present an example of using the Text Analyzer to build a CODE4 knowledge base on the subject of operating systems. The text used was assembled from articles from popular computing magazines collected on the Computer Select CD [Computer Select, 1994]. Articles were extracted from the CD-ROM that contained the keyword *operating system* plus keywords such as *Windows, DOS* and so on that refer to individual operating systems.

5.1 Preprocessing

The file, named winco94d.txt, was run through the sentence delimiter producing the file winco94d.sentences containing 14,321 sentences. This file is 1.8 Megabytes in size. It was then tagged, producing the file winco94d.tagged, 2.8 Megabytes.

The tagged file was analyzed for compound nouns. The following list shows the 20 most frequent suggested compound nouns in the file with their frequencies:

```
1. Windows NT                    653
2. Microsoft Corp                89
3. Program Manager               66
4. hard disk                     56
5. Windows applications          56
6. Microsoft Windows NT          60
7. DR DOS                        51
8. PC DOS                        51
9. Microsoft Windows             50
10. Control Panel                43
11. Advanced Server              42
12. Windows NT Advanced Server   41
13. Microsoft Corp.              40
14. new operating system         39
15. File Manager                 38
16. new process                  36
17. Brief Article                32
18. NT Advanced Server           31
```

```
19. new version                 29
20. primary thread              29
```

Figure 5.1 The 20 most frequent suggested compound nouns

Not all of these suggested compound nouns are really compound nouns. *Brief Article* is merely a header that appeared at the beginning of many of the articles in the corpus. The compounds *operating system, operating systems, disk space, Windows NT, Program Manager, Microsoft Windows NT, DR DOS, PC DOS Microsoft Windows, Control Panel, Advanced Server and Windows NT Advanced Server* were chosen by the user to be joined together by the compound noun grouping operation. The resulting file is called winco94d.taggedWithCompounds.

5.2 Statistics

The statistics operation was run on the file winco94d.tagged, although it can run equally well on the untagged file. Some of the results are shown here.

The operation found that there were 14320 sentences and broke them down by sentence length as shown in figure 5.2. The results only to length 40 are shown; the operation actually shows the results for sentences up to 100 words in length.

number of words in sentence	number of sentences	probability
1	71	0.004958
2	324	0.022626
3	190	0.013268
4	402	0.028073
5	197	0.013757
6	242	0.016899
7	328	0.022905

8	468	0.032682
9	493	0.034427
10	514	0.035894
11	493	0.034427
12	548	0.038268
13	583	0.040712
14	589	0.041131
15	602	0.042039
16	565	0.039455
17	597	0.041690
18	613	0.042807
19	554	0.038687
20	576	0.040223
21	505	0.035265
22	476	0.033240
23	453	0.031634
24	432	0.030168
25	370	0.025838
26	368	0.025698
27	339	0.023673
28	263	0.018366
29	270	0.018855
30	246	0.017179
31	182	0.012709

32	176	0.012291
33	147	0.010265
34	129	0.009008
35	131	0.009148
36	84	0.005866
37	101	0.007053
38	67	0.004679
39	80	0.005587
40	62	0.004330

Figure 5.2 Number of sentences broken down by length (number of words)

The operation counted the number of words in the text as 265,029. The results for the number of word occurrences, broken down by word length (number of characters) is shown in figure 5.3. The most common word length is 3 which is probably because of the large number of occurrences of the word *the*.

number of characters in word	number of words	probability
1	7,698	0.029046
2	40,886	0.015427
3	51,665	0.194941
4	40,899	0.154319
5	22,865	0.086274
6	22,025	0.083104
7	26,466	0.099861
8	17,055	0.064351

9	14,404	0.054349
10	7,090	0.026752
11	5,875	0.022167
12	3,944	0.014881
13	1,672	0.006309
14	821	0.003098
15	702	0.002649
16	258	0.000973
17	143	0.000540
18	112	0.000423
19	80	0.000302
>= 20	369	0.001392

Figure 5.3 Number of words, broken down by word length

5.3 Analysis

For the analysis we now had a choice of three files to work with: the sentences file, the tagged file, and the tagged file with compounds. The sentences file and the tagged file with compounds were used for most of the work. The sentences file was used for operations that required untagged text and the tagged with compounds for operations that need tagged text. (The tagged with compounds file can actually be used for all operations, if desired, since the concordance file can be used to strip the tags of sentences before putting the results through another operation.) The tagged file with compounds was only used if we wanted to *ungroup* compounds that had been grouped, to examine singly the words that make up the compounds.

5.3.1 Frequency operations

The first operation to be run on the text was the word frequency operation to search for possible terms. Below are the results of running this on the tagged with compounds file with stop words ignored:

```
1. Windows            1232
2. NT                 1079
3. Microsoft          1066
4. applications       917
5. system             914
6. file               816
7. Windows=NT         801
8. DOS                791
9. operating=system   732
10. OS                719
11. users             653
12. software          581
13. support           579
16. process           560
17. application       541
18. Unix              536
19. IBM               487
20. version           455
```

Figure 5.4 The 20 most frequent words with stop words ignored

We also looked at the word pairs and word triples of the sentences file (without compounds grouped) so that we could pick up on possible compound terms not identified by the compound finding operation. The word pairs were:

```
1. Windows NT          966
2. operating system    763
3. operating systems   178
4. Windows 3.1         177
5. Advanced Server     159
```

```
6. Microsoft Windows      157
7. DOS 6.0                135
8. Microsoft Corp         135
9. hard disk              109
10. OS 2.1                106
11. user interface        106
12. System 7              94
13. NT Advanced           90
14. MS-DOS 6.0            89
15. Control Panel         84
16. Program Manager       83
17. Windows applications  82
18. NT operating          79
19. Project DOE           57
20. PC DOS                56
```

Figure 5.5 The 20 most frequent word pairs

More instances of variations of *operating system* appear in these results than in the suggested compound nouns because some of these word pairs are in fact part of longer compounds such as *32-bit operating system.* Also, some of the pairs Windows *NT* may be part of *Microsoft Windows NT.*

In this list we find several names of operating systems: *Windows NT, DOS 6.0 . OS 2.1, System 7,* and *Windows 3.1.* These are definitely terms that should be investigated. We also see several two-word terms such as *hard disk, Control Panel,* and *user interface.*

Now let's look at the word triples from the same file:

```
1. NT Advanced Server          89
2. Microsoft Windows NT        86
3. Windows NT operating        74
4. NT operating system         73
5. Windows NT Advanced         49
6. PC DOS 6.1                  39
7. graphical user interface    31
8. 32-bit operating system     30
```

```
9. Digital Equipment Corp          25
10. Open Software Foundation       23
11. Microsoft Corp Windows         23
12. DOE Developer Release          22
13. 6.0 operating system           21
14. network operating system       19
15. SCO Open Desktop               19
16. Corp Windows NT                19
17. MS-DOS 6.0 operating           17
18. Project DOE Developer          16
19. Novell DOS 7.0                 16
20. 2.1 operating system           16
```

Figure 5.6 The 20 most frequent word triples

The operation has found terms such as *graphical user interface* and *32-bit operating system* but many of the others are simply parts of longer terms: *Windows NT Advanced,* and *NT Advanced Server,* are both probably part of the longer phrase*Windows NT Advanced Server.*

We also generated a list of all the verbs that occurred 70 or more times each, sorted alphabetically, and sorted by frequency:

access	73	installed	82
add	102	let	73
added	71	lets	75
allow	113	makes	88
allows	130	move	73
announced	96	need	176
based	157	needs	85
call	75	offers	98
called	167	provide	177
calls	90	provides	197
change	80	read	70
comes	73	require	82
contains	70	requires	76
create	191	run	391

created	106	runs	104
designed	88	select	74
distributed	75	set	175
don	152	ship	92
file	170	support	117
find	107	supports	110
found	92	take	131
get	191	want	208
help	76	won	94
include	170	work	98
includes	158	write	112
install	72		

Figure 5.7 Verbs that occurred 70 or more times sorted alphabetically ignoring stop words

run	391	announced	96
want	208	won	94
provides	197	found	92
create	191	ship	92
get	191	calls	90
provide	177	designed	88
need	176	makes	88
set	175	needs	85
file	170	installed	82
include	170	required	82
called	167	change	80
includes	158	help	76
based	157	required	76
don	152	call	75
take	131	distributed	75
allows	130	lets	75
support	117	select	74
allow	113	access	73
write	112	comes	73
supports	110	let	73
find	107	mover	73
created	106	install	72
runs	104	added	71
add	102	contains	70

offers	98	read	70
work	98		

Figure 5.8 Verbs that occurred 70 or more times sorted by frequency ignoring stop words

5.3.2 Choice of terms

Looking at these terms it was decided to create a knowledge base about the different types of operating systems (such as 32-bit operating system) and their properties, and the various operating systems themselves (such as Unix, Windows etc.) and their properties. In addition, information was also gathered on the manufacturers of operating systems. The knowledge base built containing this information could be a useful resource for answering such questions as *What are the differences between Unix and VMS?* or *What operating systems are both 32-bit and object-oriented?*

The terms that we concentrated on were those describing types of operating systems and the names of actual systems. We started with the most general term *operating system* and using the concordance operation and the super and sub concepts operation, found the types of operating systems and the names of particular systems and sketched in the basic structure of the knowledge base.

As an example of this process, we will look at how the terms *operating system* and *Windows NT* were analyzed.

Two text analyzers were kept open at the same time, one on the sentences file and one on the tagged file with compounds. Since both *operating system* and Windows *NT* were one of the compounds joined together, we could treat them as single terms in the tagged with compounds file by using *operating=system* and *Windows=NT* as the search words.

First we wanted to find out what kinds of operating systems there are. Performing a KWIC concordance on *operating=system* with the output sorted by the previous word found about 900 occurrences. By manually scanning the sorted previous words we can

quickly see which ones occur frequently before *operating=system*, modifying it. These often indicated a type of operating system. The following table shows some of the possible types of operating systems found with the number of occurrences:

new operating=system	49
Windows-NT operating=system	45
32-bit operating=system	39
Microsoft=Windows=NT operating=system	23
network operating=system	23
MS-DOS operating=system	17
Unix operating=system	15
OS 2.1 operating=system	12
Macintosh operating=system	11
desktop operating=system	10
multitasking operating=system	10
server operating=system	9
Solaris operating=system	9
high-end operating=system	5
object-oriented operating=system	5
UNIX operating=system	5
graphical operating=system	4
NeXTSTEP operating=system	2
VMS operating=system	2
16-bit operating=system	1

Figure 5.9 Possible types of operating systems found by scanning the output of a concordance on *operating=system*

It was not entirely clear which of these were really types of operating systems. Obviously a *new operating system* is not but is a *graphical operating system* a valid type of operating system or is it simply a description of one of its features? And is the *Microsoft Windows NT operating system* a sub-type of the *Windows NT operating system* or are they synonyms or unrelated concepts? These questions were answered by examining the content of the sentences in the concordance further, looking for statements that answered these questions, and by looking at the larger contexts of the sentences.

Another way to find out the types of operating systems is to look at the compounds including the pattern *operating=system**. The results of this operation, showing only compounds that occurred at least twice, are shown below:

More than 10 occurrences:

```
new operating=system                          37
Windows=NT operating=system                   27
Microsoft=Windows=NT operating=system         17
32-bit operating=system                       12
```

More than 4 and less than 10 occurrences:

```
Macintosh operating=system                    9
other operating=systems                       6
Windows operating=system                      5
new operating=systems                         5
Unix operating=system                         5
powerful operating=system                     5
```

4 occurrences:

```
dominant operating=system       operating=system kernel
Solaris operating=system        network operating=system
operating=system upgrade        server operating=system
first operating=system          32-bit operating=systems
different operating=systems
```

3 occurrences:

```
desktop operating=systems             UNIX* operating=system
complete operating=system             Unix operating=systems
Windows NT operating=systems          DOS operating=system
network operating=system market       Mac operating=system
object-oriented operating=system
```

2 occurrences:

```
high-end operating=systems            Coherent operating=system
```

```
OpenVMS AXP operating=system          MS-DOS operating=system
32-bit operating=system market        network operating=system
current Solaris operating=system      VMS operating=system
first PC-based operating=system       best operating=system
LAN operating=system environment      NT operating=system
Windows NT 32-bit operating=system    Newton operating=system
Solaris operating=system features     desktop operating=system
next-generation operating=system      operating=system market
new Windows NT operating=system       base operating=system
well-designed operating=system        true operating=system
Unix System V operating=system        operating=system wars
operating=system development      true 32-bit operating=system
32-bit operating=systems such     forthcoming operating=system
other 32-bit operating=systems    operating=system environment
operating=systems software subsidiary
```

Figure 5.10 Compounds including the pattern *operating=system**

Once we had decided, with the aid of the concordancer, which types of operating systems we would include in the knowledge base they were put in a flat list under operating system. Further analysis would show which of the named systems came under *32-bit operating system*, *object-oriented operating=system* or other categories. It would also help to detect errors in the original structure of the knowledge base: eventually, two terms that had been added as separate operating systems were found to be synonyms for the same system, and one system was found to not be an operating system at all.

Part of the hierarchy is shown below:

operating system
- **network operating system**
- **object oriented operating system**
- **16-bit operating system**
- **32-bit operating system**
- **64-bit operating system**
- **multitasking operating system**
- **DOS 6.0**
- **OpenVMS AXP**

System 7
System 7.1
Solaris
Solaris 2.1
Cairo
OS/2
Windows NT
Unix

... and so on

Figure 5.11 The initial hierarchy of operating systems

5.3.3 Conceptual analysis

Now it was time to start a more detailed analysis of the concepts chosen to be in the knowledge base to discover their properties. Starting with *operating system*, we used the find parts and find functionality operations, as well as the concordancer, to decide on an initial list of properties. These properties were things mentioned in the text that seemed to be important or interesting. At this point the list was simply a sketch which could be added to or altered later.

properties
- **made by:**
- **price:**
- **runs on:**
- **parts:**
  - **kernel:**
- **features:**
  - **multitasking:**
  - **multi-user:**
  - **multithreaded:**
  - **multiprocessing:**

Figure 5.12 The initial hierarchy of properties of *operating system*

All these properties would be inherited by the types and brands of operating systems that were subconcepts of *operating system*.

Next we did a detailed examination of each of the operating systems and types of operating systems. As an example, we will show how the analysis of one of these, Windows NT, proceeded.

At this point we believed that *Windows=NT* was an operating system but we wanted more proof. We also wanted to know what kind of operating system it is so we performed a KWIC concordance on the tagged file and requested the output sorted by the following word. This returned about 800 instances of *Windows=NT* including 47 occurrences of *Windows=NT operating=system* and its variations. This seemed to be pretty clear evidence that *Windows NT* is an operating system but we were not sure what kind of operating system it is and looking through 800 instances would take some time. If the term was not so common in the text, it would be feasible to read all the results of a concordance but we wanted something easier.

The super and sub concepts operation provided more concise information on this point, finding the following 23 sentence fragments:

```
 1182.         Windows NT  is a secure system, meaning that each user mus
 1203.  here,  Windows NT  is a graphical environment, not a graphical in
 3752.         Windows NT  is a true 32-bit operating system that uses a
 4514.  a few  Windows NT  function calls involved, but Microsoft has als
 5346.  osoft  Windows NT  is an operating system with many possibilities
 5375.         Windows NT  is a brand-new operating system that completel
 5403.  Boost  Windows NT  is a true 32-bit preemptive multitasking, mult
 5468.  uding  Windows NT  Advanced Server's sophisticated network manage
 5579.  ions,  Windows NT  is an attractive alternative to the DOS/Window
 6497.  aited  Windows NT  is a logical evolutionary step for current Win
 7088.  ative  Windows NT  sample, the Control Panel interface has change
 8578.  oft's  Windows NT  and IBM's OS/2 and new microcomputer platforms
 8586.  ch as  Windows NT  and OS/2.
 8728.  tween  Windows NT  and other Microsoft operating environments is
 9448.         Windows NT  and other 32-bit operating systems should have
 9455.  cause  Windows NT  is a new operating system, not all of our vend
 9790.   4.0,  Windows NT  is a ".0" release, no matter how Microsoft num
 9795.  hand,  Windows NT  is an efficient system and provides acceptable
10262.  oft's  Windows NT  is an extraordinary example of a rare phenomen
10301.   $495  Windows NT  is a multiplatform, 32-bit operating system pa
10725.         Windows NT  is a true 32-bit operating system, designed to
10920.  oming  Windows NT  is a true operating system designed for client
13337.  orp's  Windows NT  is a powerful operating system that uses the p
```

Figure 5.13 Possible super and subconcepts of *Windows NT*

The are five occurrences indicating that *Windows=NT* is a *32-bit operating system.* We also learned that *Windows=NT* is new and that is made by Microsoft, an indication that *Windows=NT* is probably a synonym of *Microsoft=Windows=NT.* Sentence 5403 told us that Windows NT has preemptive multitasking so *preemptive* was added as the value for the property *multitasking,* and sentence 5403 was added as the knowledge reference for this statement. Sentence 10262 is interesting: it is a perfectly well-formed ISA statement but it indicates that *Windows NT is an extraordinary example of a rare phenomenon* . Since the evidence for Windows NT being an operating system rather than a rare phenomenon is 10 to 1, it can be assumed that *operating system* is the true superconcept of *Windows NT.*

At this point we have the following hierarchy of concepts including*Windows NT*:

operating system
32-bit operating system
Windows NT

Figure 5.14 Concept hierarchy including Windows NT

And the properties of Windows NT are:

properties:
made by: Microsoft
price:
runs on:
parts:
kernel:
features:
multitasking:
preemptive multitasking:
multi-user:
multithreaded:
multiprocessing:

Figure 5.15 The second version of the property hierarchy of Windows NT

Next we performed the compounds including *Windows=NT* operation. The results provided us with various interesting compounds that could be parts or attributes of Windows NT or concepts in themselves. These would be further investigated using the concordancer.

```
Windows=NT operating system          27
Windows=NT objects                   8
Windows=NT versions                  4
```

Windows=NT object	4
Windows=NT applications	4
Windows=NT operating systems	3
Windows=NT kernel	3
Windows=NT system	3
Windows=NT System directory	3
Windows=NT environment	2
Windows=NT products	2
Windows=NT clients	2
Windows=NT systems	2
Windows=NT ships	2
Windows=NT 32-bit operating system	2
new Windows=NT operating system	2
Windows=NT implementation	2
Windows=NT version	2
Windows=NT Registry	2
Windows=NT platform	2
Windows=NT support	2
Windows=NT machines	2

Figure 5.16 Compounds including *Windows=NT*

To look for collocations with Windows NT we ran the one-word neighbourhood operation which gave the following results. First the nouns:

NOUNS	TOTAL FREQ.	-5	-4	-3	-2	-1	1	2	3	4	5
Microsoft	88	7	11	28	21	0	0	10	5	4	2
operating=system	62	0	0	0	0	0	42	2	1	9	8
Windows	35	10	5	5	5	0	0	1	3	2	4
process	32	2	4	5	2	0	0	1	8	4	6
it	30	1	3	5	2	0	1	6	8	3	1
*	26	1	2	0	2	12	1	2	3	2	1
version	26	1	1	3	16	0	2	0	0	3	0
applications	25	3	4	4	1	0	4	0	3	2	4
NT	25	7	3	7	1	0	0	1	3	0	3
you	24	2	4	4	1	1	0	7	2	1	2
Corp	23	1	2	4	12	0	0	0	1	1	2

system	21	0	1	1	2	0	3	4	3	6	1
support	18	1	1	3	4	0	2	2	1	4	0
thread	17	1	4	1	0	0	0	0	1	6	4
object	14	1	0	2	1	0	6	0	1	2	1
systems	13	1	1	2	1	0	3	1	2	1	1
versions	13	1	3	1	4	0	4	0	0	0	0

Figure 5.17 Nouns from the one word neighbourhood operation for *Windows NT*

No words stand out here as collocations quite as obviously as we saw in the examples from the optical storage media corpus A probable collocation is *operating=system* which occurs 42 times in the position directly after *Windows=NT* as in *Windows=NT operating=system*. Note that *operating=system* does not occur at all before *Windows=NT*. The word *version* occurs frequently in the -2 position and is probably part of the collocation *version of Windows=NT*. *Microsoft* also occurs frequently, a total of 88 times, but its occurrences are spread out over all word positions relative to *Windows=NT*. It is a word that is associated with *Windows=NT* (like *doctor* and *hospital* are associated) but the two don't form a collocation like *Windows=NT operating=system*. The tagger has mistagged some quotation marks as a noun which is why it appears here in the noun list. *Version* appears frequently in the -2 position; this is probably part of *version of Windows=NT*. *Corp* also appears frequently in this position and is probably part of the phrase *Microsoft Corp 's Windows=NT* (the tagger separates the *'s* from *Corp*).

Next we examined the verbs that occur with Windows NT:

VERBS	TOTAL FREQ.	-5	-4	-3	-2	-1	1	2	3	4	5
is	118	6	6	18	5	3	56	11	7	5	1
be	37	4	0	1	1	2	0	7	8	10	4
are	32	3	4	7	1	1	3	2	7	3	1
has	23	3	2	0	1	0	10	2	2	2	1
have	13	0	2	1	0	1	2	1	1	3	2
run	13	0	1	2	5	2	0	1	1	1	0
said	11	0	2	0	1	0	0	4	3	0	1
does	11	0	0	0	1	1	6	3	0	0	0

supports	11	1	0	0	0	0	7	1	1	1	0
provides	10	0	0	0	0	0	6	3	0	1	0
runs	9	1	1	0	0	0	5	2	0	0	0
includes	9	0	0	0	1	0	6	0	0	2	0
been	9	1	1	2	0	0	0	4	0	0	1
running	9	0	2	0	0	4	1	1	0	1	0
uses	8	0	0	0	0	0	6	2	0	0	0
designed	8	0	0	0	0	0	0	2	0	3	3
creates	7	0	0	0	0	0	4	2	1	0	0

Figure 5.18 Verbs from the one word neighbourhood operation for *Windows NT*

The five most common verbs are variations of the verbs *to be* and *to have* which don't convey much information by themselves. An improvement to this operation would be to let the user choose to ignore common verbs. *Run*, *runs* and *running* occur frequently but spread out over all the positions. Another improvement would be to combine the inflected forms of a verb as one root form. *Run*, *runs* and *running* would then appear together as simply *run*. The verbs that occur commonly in the +1 position and infrequently elsewhere are *supports, provides, includes* and *uses*, the verbs that indicate parts and functionality in the conceptual operations.

Finally, we looked at the adjectives that occur with Windows NT.

ADJECTIVES	TOTAL FREQ.	-5	-4	-3	-2	-1	1	2	3	4	5
new	25	4	2	0	0	3	0	0	9	2	5
32-bit	21	2	0	0	1	2	2	3	5	3	3
first	13	0	2	2	0	1	0	2	3	3	0
more	10	3	1	0	0	1	0	3	1	1	0
other	10	1	0	3	0	2	0	3	0	1	0
available	9	1	0	0	2	0	0	1	2	0	3
such	8	1	1	0	3	0	0	1	0	1	1
powerful	8	1	1	0	1	0	0	0	4	0	1
different	7	1	0	0	0	0	0	0	1	2	3
many	7	1	0	1	0	0	0	2	1	0	2
same	7	2	0	2	0	0	0	0	0	0	3
single	6	1	0	0	0	1	0	0	2	1	1

preemptive	6	0	0	1	0	0	0	1	2	0	2
small	6	0	0	0	0	0	0	0	2	1	3
long-awaited	5	0	0	0	0	5	0	0	0	0	0
forthcoming	5	1	0	0	1	3	0	0	0	0	0
important	5	0	1	2	0	1	0	0	0	1	0

Figure 5.19 Adjectives from the one word neighbourhood operation for *Windows NT*

There is only one adjective that stands out as occurring more frequently in one position: *long-awaited* in the -1 position. *New* and *forthcoming* are also frequent, indicating that *Windows=NT* is a new product. Adjectives like *more, other, such, many* and *same* tell us nothing but *32-bit, powerful, preemptive* and *important* may give us clues to follow up on. Why is Windows NT powerful and important?

To find out more about the properties of *Windows=NT* we used the find parts and find functionality operations. The find parts operation returned 50 sentences as a KWIC display from which we found the following information about *Windows=NT*:

It has a graphical user interface
It includes basic peer-to-peer networking services
It has 32-bit capability
It supports symmetric multiprocessing
It has built-in networking features
It includes Class 2 level government security standards
It includes built-in tape backup operations
It uses preemptive multitasking
It has a Boot Loader, a Control Panel, and a Cache Manager
It has a POSIX environment subsystem
It has something called the Win32 Software Development Kit

The find functionality operation returned 65 occurrences and we found the following additional information:

It supports four priority classes: IDLE, NORMAL, HIGH, REALTIME
It supports multithreaded applications
It uses the Windows 3.1 interface
It has two versions: desktop and Advanced Server
It supports 650 printers and 50 SCSI peripheral devices
It has support for uninterruptible power supplies

Its kernel provides fault-tolerance functions
It has something called the Windows NT Registry

This information was added to the knowledge base as properties and values of *Windows NT*. The properties now looked like this:

properties:
made by: Microsoft
price:
runs on:
versions: one of (desktop, Advanced Server)
parts:
kernel:
user interface: graphical, same as Windows 3.1
features:
multitasking:
preemptive multitasking:
multi-user:
multithreaded: yes
multiprocessing: symmetric
networking: built-in basic peer-to-peer networking services
security standards: Class2 level government
backup: built-in tape backup operations
number of printers supported: 650
number of SCSI peripheral devices supported: 50
support for uninterruptible power supplies:
fault-tolerance functions: supported by kernel
priority classes: IDLE, NORMAL, HIGH, REALTIME
Boot Loader:
Control Panel:

Cache Manager:
POSIX environment subsystem:
Win32 Software Development Kit:
Windows NT Registry:

Figure 5.20 The third version of the property hierarchy for *Windows NT*

Next we ran the verbs following word operation. The verbs that occurred more than twice and their frequencies were as follows:

supports	8
provides	8
uses	8
creates	7
runs	7
includes	6
assigns	4
makes	4
represents	3
free	3
offers	3
support	3
allows	3
file	3
sees	3
launch	3

Figure 5.21 The verbs following Windows *NT* that occurred 3 or more times

From the sentences that contained these verbs we learned the following about *Windows=NT*:

It lets users run existing MS-DOS and Windows applications
It assigns CPU's to threads
It creates processes
It creates a thread every time a process is initialized
It has a transaction-based recoverable file system

It frees processes and threads
Its peer-to-peer network services support file copying, e-mail, and device sharing
It is a superset of Windows 3.1 and Windows for Workgroups
It provides processor scalability
It provides crash-protected support and integrated security
It provides domain administration and management features
It provides RAID 1 and 5 fault tolerance
It runs on Intel's 386,486, Pentium and other 586 systems, MIPS R4800 and the DEC alpha RISC processor
It schedules time slices to all active threads as it runs

The verbs following word – to have found 14 occurrences including the following information about Windows NT:

It has been tested and certified on more than 1,000 Intel and RISC platforms
It has threads, semaphores, and critical sections
It has a deserialized message queue
It has a function that allows one thread to kill another thread in the same process
It has thread local storage (TLS)

The verbs following word – to be operation found 79 occurrences including the following information about Windows NT

Windows NT and Unix are high-end operating systems
It requires significant resources to run
It is due in late summer 1994
It is designed to function as a standalone, prioritized multitasking system
Its GUI is an integral part of the OS
Its core is a microkernel
It is as close to an object-oriented operating system as is currently available
It is a secure system; each user must log on to the system and establish an ID
It uses a flat memory model
It is a preemptively multitasking 32-bit version of Windows
Its backup utility is fully graphical
Its built-in networking is intended to connect small work groups and users at the departmental level
NT stands for New Technology
It is multi-user
It is an equivalent operating system to UNIX
It is unable to read partition compressed by DOS 6.0's DoubleSpace
It is designed to be almost entirely hackproof
It is constructed to serve client LAN's

The adjectives preceding word operation found the following adjectives with three or more occurrences but we learned no new information from it.

long-awaited	5
other	5
forthcoming	4
new	3
same	3
ready	3
important	3
such	3
available	3
32-bit	3
first	3

Figure 5.22 The adjectives preceding *Windows NT* that occurred 3 or more times

Additional information or clarification of the information found was obtained by examining the output of the concordance operations. From the knowledge gained by running these various operations we were able to create the property hierarchy shown in Figure 5.24 below

properties:

made by: Microsoft
price: $495
runs on: all of (Intel's 386,486, Pentium and other 586 systems, MIPS R4800, DEC alpha RISC processor)
versions: one of (desktop, Advanced Server)
superset of: both of (Windows 3.1, Windows for Workgroups)
tested and certified on: more than 1,000 Intel and RISC platforms
hackproof: designed to be almost entirely hackproof
equivalent to: Unix
name acronym for: Windows New Technology
projected release date: late summer 1994

parts:
- **kernel:** microkernel
- **user interface:** graphical, same as Windows 3.1
- **hardware application layers:**
- **POSIX environment subsystem:**

features:
- **multitasking:**
  - **preemptive multitasking:**
- **multi-user:** yes
- **multithreaded:** yes
- **multiprocessing:** symmetric
- **networking:** built-in basic peer-to-peer networking services
- **network services support:** all of (file copying, e-mail, device sharing, small work groups and users at the departmental level)
- **security standards:** Class2 level government, each user must log on to the system and establish an ID
- **backup:** built-in tape backup operations, graphical
- **number of printers supported:** 650
- **number of SCSI peripheral devices supported:** 50
- **support for uninterruptible power supplies:**
- **fault-tolerance:** supported by kernel
- **fault tolerance level:** RAID 1 and 5
- **applications run:** MS-DOS, Windows
- **file system:** transaction-based recoverable file system
- **domain administration and management features:**
- **serves:** client LAN's

creates: processes, threads
frees: processes, threads
assigns: threads to CPU's
schedules: time slices to all active threads as it runs
semaphores:
critical sections:
deserialized message queue:
priority classes: IDLE, NORMAL, HIGH, REALTIME
thread local storage:

memory model: flat
processor scalability:
user and group types: all of (Guests, Users, Backup Operators, Power Users, Administrators)
crash-protected support:
Boot Loader:
Control Panel:
Cache Manager:
Win32 Software Development Kit:
Windows NT Registry:
deficiencies: unable to read partition compressed by DOS 6.0's DoubleSpace

Figure 5.23 The fourth version of the property hierarchy for *Windows NT*

This knowledge base was put together the author, a user without an extensive understanding of the different kinds of operating systems and their features but with a good understanding of the English language. The section on Windows NT was generated with 2 to 4 hours of work, including preprocessing the file. If done manually, it could take this long to simply read the text through once, let alone understand it and create a consistent knowledge base.

This knowledge base probably contains mistakes but it could be quickly scanned and corrected by an expert in the field. The Text Analyzer thus allows a generalist, fluent in English, to create knowledge bases from text without a full understanding of the subject, that can be verified by a specialist. The specialist would probably not have the time (or the interest) in creating the entire knowledge base from his own knowledge or from texts.

6 Summary, discussion and future work

This chapter summarizes the contributions of the thesis and presents some ideas for future work on the Text Analyzer.

6.1 Summary

The main goal of thesis was to develop a tool that would aid people in the task of knowledge acquisition from texts. We began in Chapter 1 by examining the problems of finding knowledge in natural language documents – why it is necessary and why it is difficult to do – and suggesting how a computerized tool could make this process easier.

In Chapter 2 we reviewed some research that relates to this work, first in the field of linguistics, next in the area of knowledge management systems, and thirdly in the very new field that combines methods from both these areas.

Chapter 3 introduces the tool, the Text Analyzer, and explains all the operations it performs with examples.

Chapter 4 discusses the implementation of the Text Analyzer – the use of Icon and C for the actual text processing and Smalltalk for the user interface, some techniques for fast text searching, and a discussion of why indexing and parsing were not used in this project.

In Chapter 5 we present an actual example of building a CODE4 knowledge base from a text on the subject of operating systems using the Text Analyzer.

In this chapter, Chapter 6, we also present some future work to be done on the Text Analyzer.

6.2 Discussion

After designing and implementing the Text Analyzer we have come to the following conclusions:

- The Text Analyzer represents a new kind of tool with a new way of using text. We think this is valuable as a way of saving time when searching for information in documents, as a way of verifying documents for consistency and clarity of writing, and as an aid for linguistic analysis.

- We would like to see more research done that meshes linguistics with knowledge acquisition. We believe that our research and that of others such as Mikeev and Finch show that this combination of techniques has produced new and useful methods for analyzing text and has opened up a whole new area of research. The operations provided by the Text Analyzer are only the beginning of what could be done – already linguists who are using the Text Analyzer in their research have suggested more operations that would help them in their work. The Text Analyzer provides the basic structure in which to test new operations.

- Tools like the Text Analyzer can be a step between written texts and knowledge bases. Although the average person may be intimidated by the idea of having to create a knowledge base, he may be more comfortable using the Text Analyzer or a simlar tool to extract knowledge from texts. It is an easy step, then, to put the extracted knowledge into a knowledge base, especially if it is directly connected like CODE4 is to the Text Analyzer. This two-step process of extracting knowledge using the Text Analyzer and *then* putting it into a knowledge base is less intellectually taxing than going straight from a text to a knowledge base.

6.3 Future Work

There is much interesting work that could be done to improve the Text Analyzer and extend its functions. As a start, we would like to see the following enhancements made to the Text Analyzer:

- Stemming. This would be very valuable for many of the Text Analyzer's operations. This would be particularly important for any of the operations that present words by frequency where occurrences of the same word with different inflections are now counted as different words.
- Giving the user the option of ignoring the stop words in more of the operations.
- Converting the remaining Icon operations to C.
- Finding the direct objects of the verbs after the search word. Some work has already been done on this by the author.
- Discovering more *knowledge probes* for conceptual operations.
- Sorting concordance lines generated by the conceptual operations by following (or previous) word and then sorting them again by the frequency of the following (or previous) word. This gives the parts, for example, of the search word arranged by the amount of evidence supporting them (the number of sentences in which they appear).
- Finding sentences that contain important information but which do not contain the search word. These are often the sentences immediately after a sentence containing the search word but where the search word is replaced by a pronoun.
- Automatically writing the sentence number(s) to the knowledge reference facet in CODE4. Making a hypertext link from the knowledge reference in the CODE4 knowledge base to the sentence in the text where the word occurred.
- Devising a method to mark paragraphs, chapters etc. with the sentence delimiter program. This would allow the programs to search for information by paragraph, chapter etc. rather than just within a sentence.

- Improving the user interface. The emphasis of this research was on the operations of the Text analyzer rather than the user interface.

Epilogue

After this thesis was completed, it seemed a good idea to check it for consistency and clarity using the Text Analyzer, since this is one of the uses the Text Analyzer was designed for.

The thesis was saved from Microsoft Word as text only, and a Text Analyzer was opened on this file. After sentence delimiting, we ran the word frequency, word pair frequency and word triple frequency operations out of interest (the most common word pair was *Text Analyzer* which occurred 76 times). The statistics operation revealed that there are 1099 sentences containing 26,816 words in the thesis (an average of 24 words per sentence).

The previous operations were done mostly for interest but the concordance operation was used to check the clarity and consistency of the language used in the thesis. For example, a concordance was done on the word *concordance* to make sure that the first time it was used, it was defined adequately, and to verify that it was used consistently throughout the thesis. This kind of check could have been done using a word processing operation but it is easier and faster to do using the Text Analyzer and it is helpful to have all the sentences containing the search word grouped on one screen.

Appendix A – Penn Treebank Part of Speech Tags

1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition/subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative

23.	RP	Particle
24.	SYM	Symbol (mathematical or scientific)
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund/present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd-person singular present
33.	WDT	*wh*-determiner
34.	WP	*wh*-pronoun
35.	WP$	Possessive *wh*-pronoun
36.	WRB	*wh*-adverb
37.	#	Pound sign
38.	$	Dollar sign
39.	.	Sentence - final punctuation
40.	,	Comma
41.	:	Colon, semi-colon
42.	(	Left bracket character
43.	)	Right bracket character
44.	"	Straight double quote
45.	`	Left open single quote
46.	``	Left open double quote
47.	'	Right close single quote

48. ” Right close double quote

Appendix B – Stop Words

```
a
about
after
all
also
although
an
and
any
April
are
as
at
August
Aug
be
because
become
becomes
becoming
been
before
being
between
both
but
by
can
can't
cannot
could
couldn't
December
Dec
did
do
does
doesn't
don't
each
eight
February
Feb
first
few
five
for
four
from
go
had
has
have
having
he
her
him
his
how
I
if
in
into
is
isn't
it
its
it's
January
Jan
July
June
know
like
man
made
make
March
Mar
May
may
me
might
month
months
more
most
must
my
n
new
nine
no
not
November
Nov
October
Oct
of
```

```
on
one
only
or
other
our
p
Record
s
said
same
say
says
second
see
Select
September
Sept
seven
should
since
six
she
so
such
t
ten
than
that
the
The
their
them
then
these
there
they
they'll
this
those
three
third
third
those
to
To
too
two
Two
up
us
Us
use
used
uses
using
v
very
was
we
week
weeks
were
what
when
where
which
who
why
with
will
with
would
year
years
you
you'll
your
you're
1988
1989
1990
1991
1992
1993
1994
```

Appendix C – Abbreviations

A.D.
Ave.
B.C.
Bldg.
Calif.
Co.
Corp.
Cres.
Ct.
Ctr.
Dr.
ext.
Ga.
Inc.
ISO.
L.L.
Ltd.
Mass.
Md.
Minn.
M.O.S.T.
no.
No.
N.E.
N.W.
P.O.
R.I.
S.E.
S.W.
St.
Ste.
U.S.
U.S.A.
Vol.
vs.
Vt.

```
Wis.
```

References

[Ahmad et al, 1994] Khurshid Ahmad, Andrea Davies, Heather Fulford and Margaret Rogers, What is a term? The semi-automatic extraction of terms from text. In *Translation Studies: An Interdiscipline. Selected Proceedings of the Vienna Conference*, M. Snell-Hornby and F. Pochhacker (eds.), pp. 9-12, 1992.

[Ahmad and Fulford, 1992] K. Ahmad and Heather Fulford, *Semantic Relations and their Use in Elaborating Terminology*, Computing Sciences Report CS-92-7, University of Surrey, 1992

[Ahmad and Rogers, 1992a] Khurshid Ahmad and Margaret Rogers, Terminology Management: a corpus-based approach. In *Translating and the Computer 14: Quality, Standards and the Implementation of Technology*, R. Turner (ed.),ASLIB, London, 1992.

[Ahmad and Rogers, 1992b] Khurshid Ahmad and Margaret Rogers, Translation and Information Technology: The Translator's Workbench, In *RECALL* Journal of the Computers in Teaching Initiative Centre for Modern Languages, University of Hull. No. 6, pp. 3-9, May 1992.

[Aijmer and Altenberg, 1991] Karin Aijmer and Bengt Altenberg, Introduction to *English Corpus Linguistics*, Karin Aijmer and Bengt Altenberg (eds.), Longman, London, 1991

[Anjewierden et al., 1992] A. Anjewierden, J. Wielemaker and C. Toussaint, Shelley, computer-aided knowledge engineering, *Knowledge Acquisition*, 4, 1992

[Atkins, 1992] B.T.S. Atkins, Tools for computer-aided corpus lexicography: the Hector Project. *Papers in Computational Lexicography, COMPLEX '92.* , Hungarian Academy of Sciences, Budapest, 1992

[Aussenac-Gilles et al., 1995] Nathalie Aussenac-Gilles, Didier Bourigault, Anne Condamines and Cecile Gros, How Can Knowledge Acquisition Benefit from Terminology?, *9th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, 1995.

[Benson, 1990] M. Benson, Coliocations and general-purpose dictionaries, *International Journal of Lexicography*, 2,1-14, 1990.

[Black, Lafferty and Roukos, 1992] Ezra Black, John Lafferty and Salim Roukos, Development and Evaluation of a Broad-Coverage Probabilistic Grammar of English-Language Computer Manuals, *Proceedings of the Association for Computational Linguistics*, Delaware, 1992

[Bourigault, 1995] Didier Bourigault, LEXTER, a Terminology Extraction Software for Knowledge Acquisition from Texts, *9th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, 1995.

[Bourigault, 1994] Didier Bourigault, LEXTER, un Logiciel d'EXtraction de TERminologie. Application a l'acquisition des connaissances a partir de textes. Ph.D. Thesis, Ecole des Hautes Etudes en Sciences Sociales, Paris, 1994

[Bourigault and Lepine, 1994] Didier Bourigault and Pascal Lepine, Une methode d'utilisation de LEXTER en acquisition des connaissances a partir de textes. In *Proc. 5eme Journees d'Acqusition des Connaissances*, Strasbourg, 1994.

[Brill, 1992] Eric Brill, A simple rule-based part of speech tagger, *Proceedings of the Third Conference on Applied Natural Language Processing*, Trento, Italy, 1992

[Brill and Marcus 1992] Eric Brill and Mitch Marcus, Tagging an Unfamiliar Text With Minimal Human Supervision, *Proceedings of the Fall Symposium on Probabilistic Approaches to Natural Language*, AAAI, 1992

[Brill, 1993] Eric Brill, A Corpus-Based Approach to Language Learning. Ph.D. Dissertation, Department of Computer and Information Science, University of Pennsylvania, 1993

[Brill, 1994] Eric Brill, A Report of Recent Progress in Transformation-Based Error-Driven Learning, *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, WA., 1994

[Brown, Callan and Croft, 1994] Eric W. Brown, James P. Callan and W. Bruce Croft, Fast Incremental Indexing for Full-Text Information Retrieval, *Proceedings of the 20th VLDB Conference*, Santiago, Chile, 1994

[Burnard, 1992] Lou Burnard, Tools and Techniques for Computer-assisted Text Processing, In *Computers and Written Texts*, C. Butler (ed.), Blackwell, Oxford, 1992

[Computer Select, 1994] Computer Select, Computer Library, Information Access Co., New York, 1994

[Cormen et al., 1990] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, Mass., 1990

[Diederich et al., 1987] J. Diederich, I. Ruhmann and M. May, KRITON: A knowledge-acquisition tool for expert systems, *International Journal of Man-Machine Studies*, 26, 1987

[Francis and Kucera, 1982] W. Nelson Francis and Henry Kucera, *Frequency Analysis of English Usage, Lexicon and Grammar*, Houghton Mifflin Co., Boston, 1982

[Francis, 1982] W. Nelson Francis, Problems of Assembling and Computerizing Large Corpora, In *Computer Corpora in English Language Research*, Stig Johansson (ed.), Norwegian Computing Centre for the Humanities, Bergen, 1992

[Griswold and Griswold, 1990] Ralph E. Griswold and Madge T. Griswold, *The Icon Programming Language*, Prentice Hall, Englewood Cliffs, N.J., 1990

[Hearst, 1992] Marti A. Hearst, Automatic Acquisition of Hyponyms from Large Text Corpora, *Actes de COLING-92*, Nantes, 1992

[Hobbs et al., 1992] Jerry R. Hobbs, Douglas E. Appelt, John Bear, Mabry Tyson and David Magerman, Robust Processing of Real-World Natural-Language Texts, *Text-Based Intelligent Systems*, Erlbaum, Hillsdale, N.J., 1992

[Keller, 1987] Robert Keller, *Expert System Technology, Development and Application,* Yourdon Press, Englewood Cliffs, N.J., 1987

[Leech, 1991] Geoffrey Leech, The State of the Art in Corpus Linguistics, In *English Corpus Linguistics,* Karin Aijmer and Bengt Altenberg (eds.), Longman, London, 1991

[Leech and Fligelstone, 1992] Geoffrey Leech and Steven Fligelstone, Computers and Corpus Analysis, In *Computers and Written Texts*, C. Butler (ed.), Blackwell, Oxford, 1992

[Lenat and Guha, 1990] Douglas B. Lenat and R.V. Guha, *Building Large Knowledge-Based Systems* Addison-Wesley, Reading, Mass, 1990

[Lenat et al., 1990] Douglas B. Lenat, Ramanathan V. Guha, Karen Pittman, Dexter Pratt and Mary Shepherd, CYC: Toward Programs with Common Sense, *Communications of the ACM*, Vol. 33, No. 8, Aug. 1990

[Meyer, 1995] Ingrid Meyer, personal correspondence, 1995

[Meyer, Skuce, Bowker and Eck, 1992] Ingrid Meyer, Douglas Skuce, Lynne Bowker and Karen Eck, Towards a new generation of terminological resources: An experiment in building a terminological knowledge base. *13th International Conference on Computational Linguistics(COLING)*, Nantes, 1992

[Mikeev and Finch, 1995] Andrei Mikeev and Steven Finch, A Workbench for Acquisition of Ontological Knowledge from Natural Language, *9th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, 1995.

[Patten, 1992] Terry Patten, Computers and Natural Language Parsing, In *Computers and Written Texts*, C. Butler (ed.), Blackwell, Oxford, 1992

[Salton and McGill, 1983] Gerald Salton and Michael J. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill Book Company, New York, 1983.

[Sinclair, 1991] John Sinclair, *Corpus, Concordance, Collocation*, Oxford University Press, 1991.

[Skuce and Lethbridge, 1995 to appear] Doug Skuce and Timothy C. Lethbridge, CODE4: A Unified System for Managing Conceptual Knowledge, to appear in *International Journal of Human-Computer studies*]

[Smadja, 1993] Frank Smadja, Retrieving Collocations from Text: Xtract, *Computational Linguistics*, Vol. 19, No. 1, 1993.

[TACT, 1994] TACT Version 2.1 ReadMe File, 1994

[TACT, 1990] TACT Manual, 1990

[Turtle and Croft, 1992] Howard Turtle and W. Bruce Croft, A Comparison of Text Retrieval Models, *The Computer Journal*, Vol. 35, No.3, 1992, pp. 279-290