INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

NOTE TO USERS

This reproduction is the best copy available

UMI

AUTOMATIC SEMANTIC HEADER GENERATOR

Sami Samir Haddad

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

July 1998
© Sami Samir Haddad, 1998

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-39485-9

Canada

Abstract

Automatic Semantic Header Generator

Sami Samir Haddad

As the amount of information and the number of Internet users grow, the problem of indexing and retrieval of electronic information resources becomes more critical. The existing search systems tend to generate misses and false hits due to the fact that they attempt to match the specified search terms without context in the target information resource. The COncordia INdexing and DIscovery system is an indexing system. It is a powerful means of helping users locate documents, software, and other types of data among large repositories. In environments that contain many different types of data, content indexing requires type-specific processing to extract information effectively.

The Semantic Header, which is proposed by Desai [11], contains the semantic contents of information resources. It provides a useful tool in searching for a document based on a number of commonly used criteria. The information from the semantic header could be used by the search system to help locate appropriate documents with minimum effort.

This thesis introduces an automatic tool for the extraction and storage of some of the meta-information in a Semantic Header and an automatic text classification scheme.

Acknowledgments

There are many people, both personal friends and professionals from academia, to whom I owe this thesis. I would like to take this opportunity to show my appreciation to them.

First and foremost, I would like to express my gratitude to my parents, who throughout the years, have loved and nurtured me. I not only thank you for supporting me throughout my education, but I would also like to show my deepest gratitude to you for allowing me to live and to grow up in a stable and healthy environment. My love is also expressed to my brothers, who throughout my childhood and until this day, have loved and respected me.

I would like to thank my supervisor, Dr. Desai, for his guidance and for his encouragement throughout the many phases of this thesis. It is only through your proper support that I have overcome the many challenging phases of this thesis and made my thesis work a pleasant and extremely educational experience. Dr Desai, I am forever indebted to you, and I am certain that we will meet and work again in the future.

I am grateful to Lee Harris and Carol Coughlin, who have been helpful in some of my thesis work.

I would like to express my sincere recognition to Concordia University, whose excellence in academic teaching has attracted the finest professors from academia. The few years that I have spent at Concordia, have provided me with excellent training, and they have allowed me to become a well-grounded individual, who is ready to enter the professional world with great confidence.

Last and by no means least, I would like to thank my friends and my girl friend, for being there for me in time of need. Our friendship has proven to be solid over the years, and I can only hope that it will last for a lifetime.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Discovery Problem

Rapid growth in data volume, user base and data diversity render Internet-accessible information increasingly difficult to use effectively. At this time, a number of information sources, both public and private, are available on the Internet. They include text, computer programs, books, electronic journals, newspapers, organisational, local and national directories of various types, sound and voice recordings, images, video clips, scientific data, and private information services such as price lists and quotations, databases of products and services, and speciality newsletters [12]. There is a need for an automated search system that allows easy search for and access to relevant resources available on the Internet. Proper functioning of this system will require a proper indexing of the available information. Thus, secondary information called meta-information must be extracted and used as an index to the available primary resource. Building this index requires information extraction methods tailored to each specific environment. The semantics of the files in which the primary resource is stored will be exploited in order to extract and summarise the relevant information that will support the resource discovery. To do this, the primary file type should be identified and then the type specific selection and extraction methods are applied to the file.

It is envisioned that regional and/or specialised databases will be created to maintain archives of the cover pages (or Semantic Headers). These databases could be searched by cooperating distributed expert systems to help users in locating pertinent documents. Such a system is currently under development at Concordia University and is called Concordia INdexing and DIscovery system, or CINDI.

1.2 Proposed solution

CINDI, a system under development at Concordia University, provides a mechanism to register, search and manage the meta-information, with the help of an easy to use graphical user interface. This meta-information, which is described in chapter 2, is the Semantic Header, that is stored in the CINDI system. CINDI tries to avoid problems caused by differences in semantics and representation as well as incomplete and incorrect data cataloguing. It also tries to avoid the problems caused by the difference in index terms. This meta-information could be entered either by the primary resource provider or by the Automatic Semantic Header Generator (ASHG). ASHG, a software that generates some meta-information of the submitted document, assists the user in this process. This thesis introduces ASHG, which aims at saving the primary resource provider's time by automatically generating and extracting part of the meta-information (Semantic Header) of the document and classifying the resource under a list of subject headings. As the provider helps in this process by verifying and correcting the Semantic Header entry, there is the potential for its accuracy is high.

1.3 Organisation of the thesis

This thesis is organised as follows. In chapter 2, we will introduce the CINDI system. Chapter 3 describes information retrieval, its history and some of the algorithms used in that field. Automatic text retrieval, natural language processing and text classification is also discussed in chapter 3. At the end of chapter 3, we describe some retrieval and information extraction systems.

Chapter 4 covers the Thesaurus used and how it is built. Chapter 5 describes the Automatic Semantic Header Generator, or ASHG. This chapter covers the basic subparts used by it as the type recognition, and the extractors. In chapter 6, we test and compare the classification of our generated index with the ones produced by cataloguers or the document's author's opinion. Finally in chapter 7, we draw our conclusion.

Chapter 2

The CINDI system

The current practice in most research institutes, universities and business organisations to interconnect their computing facilities using a digital network is the accepted method of sharing resources. Such networks, in turn, are interconnected allowing information to be exchanged across networks using appropriate data transfer protocols.

There is a need for the development of a system which allows easy search for and access to resources available on the Internet. Solving the problem of fast, efficient and easy access to the documents can be started by building a standard index structure and building a bibliographic system using standardised control definitions and terms. Such definitions could be built into the knowledge-base of an expert system based index entry and search interface. The purpose of indices and bibliographies (secondary information) is to catalogue the primary information and allow easy access to it.

Preparing the primary source's meta or secondary information requires finding the primary source, identifying it as to its subject, title, author, keywords, abstract, etc. Since it is to be used by many users, it has to be accurate, easy to use and properly classified.

Attempts to provide easy search of relevant documents has lead to a number of systems including WAIS, and more recently a number of Spiders, Worms and other creepy crawlers [9, 20, 28, 39, 68,

60, 71, 72, 73].

However, the problem with many of these tools is that their selectivity of documents is often poor [12]. The chances of getting inappropriate documents and missing relevant information because of poor choice of search terms is large. These problems are addressed by CINDI, which provides a mechanism to register, manage and search the bibliographic information.

2.1 Overview of CINDI

The overall CINDI system uses knowledge bases and expert sub-systems to help the user in the registering and the search processes. CINDI standardises the terms. The index generation and maintenance sub-system uses CINDI's thesaurus to help the provider of the resource select correct terms for items such as subject, sub-subject and keywords. Similarly, another expert sub-system is used to help the user in the search for appropriate information resources [11].

2.1.1 The Semantic Header

For cataloguing and searching, CINDI uses a meta-data description called a Semantic Header to describe an information resource. The Semantic Header includes those elements that are most often used in the search for an information resource. Since the majority of searches begin with a title, name of the authors (70%), subject and sub-subject (50%) [27], CINDI requires the entry for these elements in the Semantic Header. Similarly, the abstract and annotations are relevant in deciding whether or not a resource is useful, so they are included too.[56, 12]. A brief descripton of the semantic header elements follows:

Title, Alt-title

The title field contains the name of the resource that is given by the creator(s). The alternate title field is used to indicate a secondary title of the resource.

Subject

The subject and sub-subjects of the resource are indicated in the next field which is a repeating group. This field contains a list of possible subject classifications of the resource.

Language, Character Set

The character set and the language are the ones used in resource.

Author and other responsible agents

The role of the person associated with the document, for instance, author, editor, and compiler. This includes fields such as name, postal address, telephone number, fax number, and email address.

Keyword

This field contains a list of keywords mentioned in the resource.

Identifier

The identifiers for the document. Example of identifiers are, ISBN(International Standard Book Number), URL (Universal Resource Locator) of the document. This is a multi-valued slot in case the document is available in many formats or is electronically stored at more than one site.

Date

The date on which the document was created, catalogued, and the date on which the document will expire, if any.

Version

The version number, and the version number being superseded, if any, are given in these elements.

Classification

The legal, security or other type of classification of the document. For each, nature of classification is specified.

Coverage

It indicates the targeted audience of the document or it may indicate cultural and temporal aspect of the document's content.

System Requirements

The document being an electronic one requires certain system requirements for it to be displayed or used. The components are the hardware, the software or the network and for each the minimum needs.

Genre

It is used to describe the physical or electronic format of the resource. It consists of a domain and the corresponding value or size of the resource.

Source and Reference

The Source indicates the documents being referenced or which were required in its preparation. It could also be the main component for which the current document is an addendum or attachment.

Cost

In case of a resource accessible for a fee, the cost of accessing it is given.

Abstract

The abstract of the document is either provided by the author or by ASHG.

Annotations

Annotations put in by readers of the document.

User ID, Password

A Provider ID of at least six characters and a password of four to eight characters. More than one semantic header by the same provider can have the same ID and password.

2.1.2 Semantic Header's Mark-up Schema

The semantic header's fields are stored using a language based on the SGML mark-up language:

```
<semhdr>

   <title> required </title>
      <alt-title> OPTIONAL </alt-title>

   <Subject> required: a list each of which includes fields
          for subject and up to two levels of sub-subject:
          at least one entry is required </Subject>

   <language> OPTIONAL: of the information resource </language>

   <char-set> OPTIONAL: character set used </char-set>

   <author> required: a list each of which includes role, name,
          organisation, address, etc. of each person/institute
          responsible for the information resource: at least the
          name or the organisation and address is required </author>

   <Keyword> required: a list of keywords </Keyword>

   <Dates>
      <Created> required: </Created>
      <Expiry> OPTIONAL: </Expiry>
      <Updated> system generated </Updated> </Dates>

   <Version> OPTIONAL: version of the resource </Version>

   <Supersedes> OPTIONAL: which version is being replaced
          </Supersedes>

   <Coverage> OPTIONAL: audience, spatial, temporal </Coverage>

   <Classification> OPTIONAL: nature (legal, security level etc.)
          of the resource </Classification>
```

```
   <Identifier> A list of domains for identifiers and the
          corresponding values: typical identifiers could
          be one of more Unique Resource Locator(URL), Call
          No. for the resource, unique name of the resource
          (URN), site where the item is to be archived: at
          least one required </Identifier>

   <Abstract> OPTIONAL but recommended </Abstract>

   <Annotation> OPTIONAL: </Annotation>

   <SysReq> OPTIONAL: list of system requirements for example
          hardware and software: the component and the corres-
          ponding requirements are given </SysReq>

   <Source> OPTIONAL: gives the source or related list of
          resources for each such resource it indicates a
          relationship and gives an identifier which includes
          the domain and the corresponding value </Source>

   <size> size of the resource in appropriate units
          (e.g., bytes) </size>

   <Cost> OPTIONAL: cost of accessing the resource </Cost>

   <control>
      <Ac> account number </Ac>
      <password> required: encoded password or digital signature of
         provider of resource for initial entry and subsequent update
         </password>
      <signature> digital signature of the resource for authentication
         </signature> </control>
</semhdr>
```

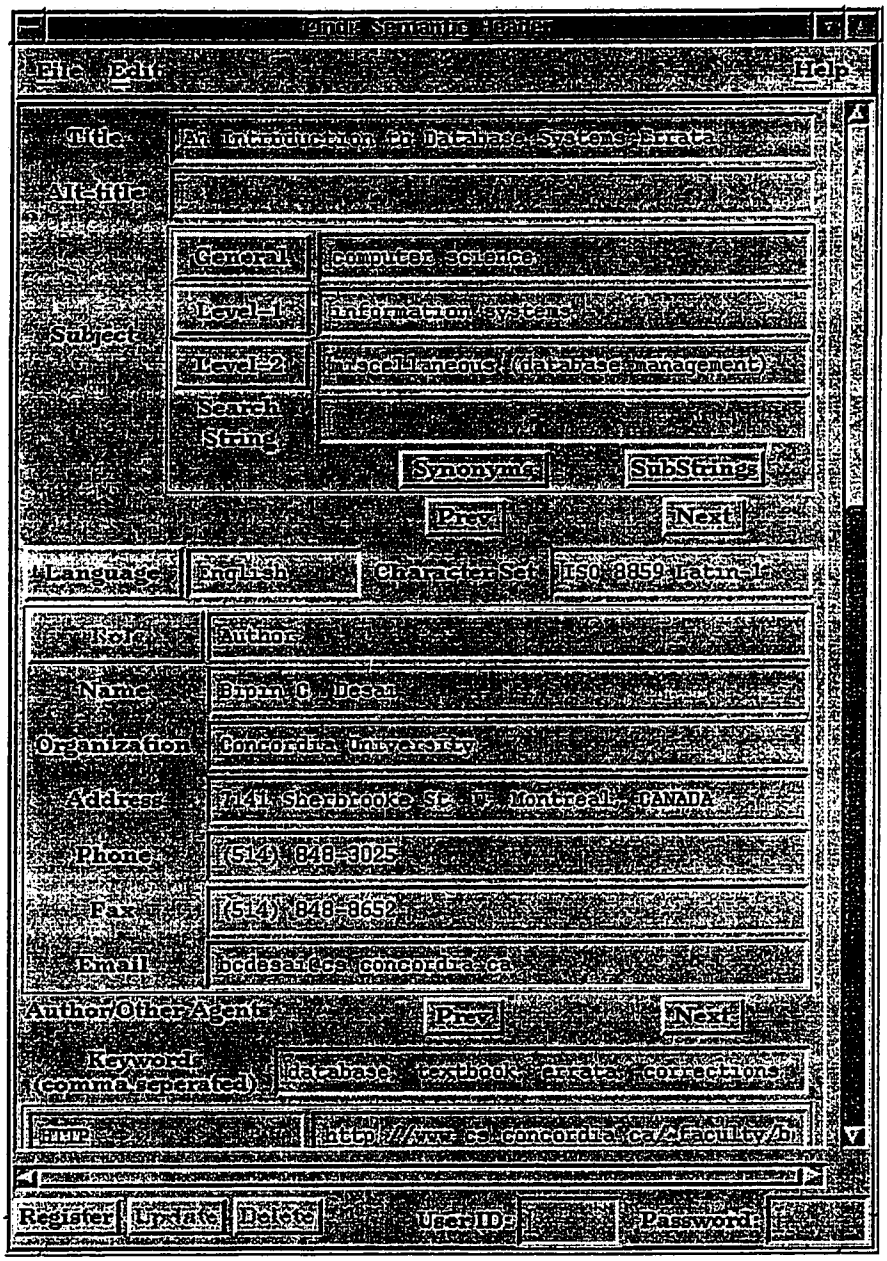

Figure 1: CINDI' Semantic Header Graphical interface_(a)

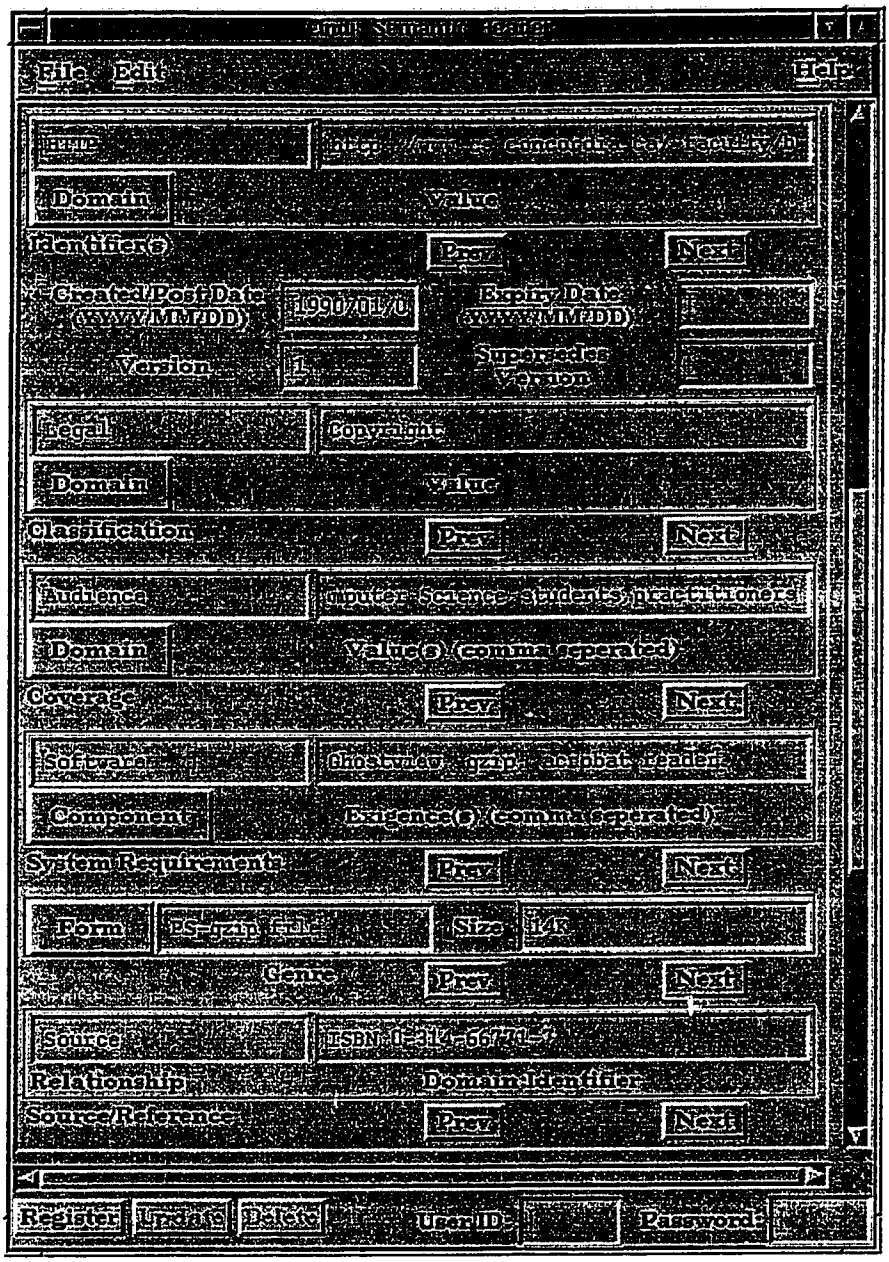

Figure 2: CINDI' Semantic Header Graphical interface_(b)

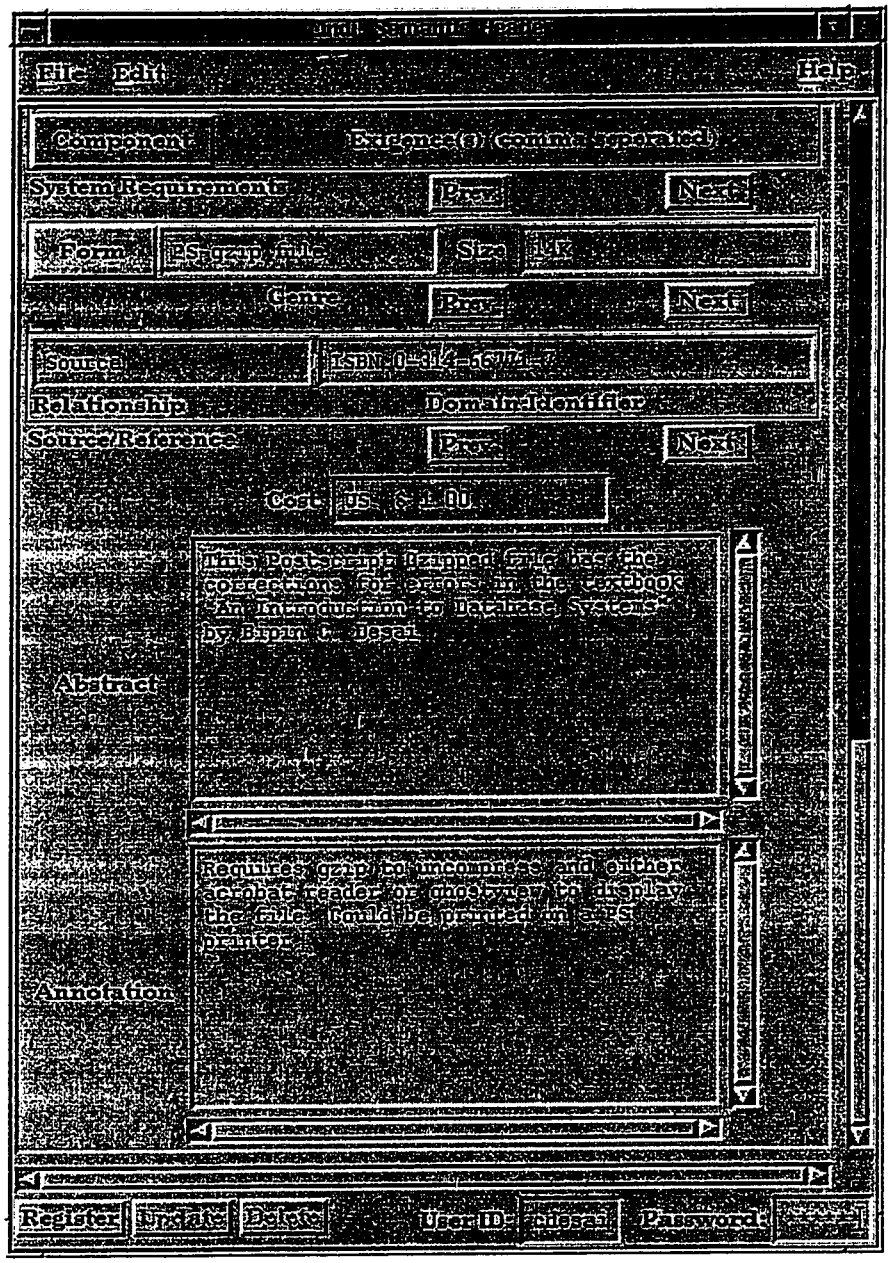

Figure 3: CINDI' Semantic Header Graphical interface_(c)

Next, the Semantic Header Database system is described.

2.1.3 The Semantic Header Database System

The index entries registered by a provider of a resource is stored in a distributed database system (SHDDB). From the point of view of the users of the system, the underlying database may be considered to be a monolithic system. In reality, it would be distributed and replicated allowing for reliable and failure-tolerant operations. The interface hides the distributed and replicated nature of the database. The distribution is based on subject areas and as such the database is considered to be horizontally partitioned [10].

It is envisaged that the database on different subjects will be maintained at different nodes of the Internet. The locations of such nodes need only be known by the intrinsic interface. A database catalog would be used to distribute this information. However, this catalog itself could be distributed and replicated as is done for distributed database systems.

The Semantic Header information entered by the provider of the resource using a graphical interface is relayed from the user's workstation by a client process to the database server process at one of the nodes of the SHDDB. The node is chosen based on its proximity to the workstation or on the subject of the index record. On receipt of the information, the server verifies the correctness and authenticity of the information and on finding everything in order, sends an acknowledgment to the client.

The server node is responsible for locating the partitions of the SHDDB where the entry should be stored and forwards the replicated information to appropriate nodes. For example, the semantic header entry would be part of the SHDDB for subjects Computer Science and Library Studies.

Similarly the database server process is responsible for providing the catalogue information for the search system. In this way the various sites of the database work in a cooperating mode to

maintain consistency of the replicated portion. The replicated nature of the database also ensures distribution of load and ensures continued access to the bibliography when one or more sites are temporarily nonfunctional.

2.1.4 The CINDI's Search System

CINDI guides the user in entering the various search items in a graphical interface similar to the one used by the index entry system. The search system also uses a graphical interface and a client process. Once the user has entered a search request, the client process communicates with the nearest SHDDB catalogue to determine the appropriate site of the SHDDB database. Subsequently, the client process communicates with this database and retrieves one or more semantic headers. The result of the query could than be collected and sent to the user's workstation. The contents of these headers are displayed, on demand, to the user who may decide to access one or more of the actual resources. It may happen that the item in question may be available from a number of sources. In such a case the best source is chosen based on optimum costs. The client process would attempt to use appropriate hardware/software to retrieve the selected resources [12].

Chapter 3

Information Retrieval

3.1 What is Information Retrieval?

Information Retrieval (IR) is concerned with the representation, storage, organisation and accessing of information. The first step in the retrieval process is for the user to state the information needed. This has to be done in a format that enables the IR system to understand it and to act on it [19]. To facilitate the task of finding items of interest, libraries and information centers provide information users with a variety of auxiliary aids. Each incoming item is analysed and appropriate descriptions are chosen to reflect the information content of the item. Retrieval effectiveness is typically measured by two metrics, precision, which is the percentage of the retrieved documents that are relevant to the information need, and recall, which is the percentage of relevant documents in the collection that are retrieved. [19].

In this chapter, we will discuss the history of information retrieval, automatic document indexing or representation, algorithms used by the IR community, natural language processing, the automatic sentence extraction and abstract selection and the text classification. We will also be portraying Salton's SMART retrieval system, Oracle's ConText, Nordic and Harvest's Essence information retrieval and extraction systems.

Indexing is the basis for retrieving documents that are relevant to the user's need [34]. Building an accurate representation of a document, which would increase precision, is one of CINDI's main concerns. Compact descriptions of a document's index may increase the efficiency of matching and the effectiveness of classifying textual material as relevant or non-relevant. Document retrieval imposes conflicting normalising and accurate demands [34]. As a result, variations in indexing that increase precision usually decrease recall, and vice versa. The fundamental goal is to increase both. There are numerous types of indexing languages. One which uses the same terms found in the document and another which is limited to those from a controlled languages [34].

3.1.1 Information Retrieval Background

Tests of indexing languages have shown that indexing documents by individual terms corresponding to words or word stems produces results that are at least as good as those produced when indexing by controlled vocabularies [34].

Luhn[36] used frequency counts of words in the document text to determine which words were sufficiently significant to represent the document. The use of statistical information about distributions of words in documents was further exploited by Maron and Kuhn [37] and Stiles [58] who obtained statistical associations between keywords.

Statistical Document Retrieval methods assign higher numeric weights to terms showing evidence of being good content indicators, causing them to have greater influence on the ranking of the documents. The number of occurrences of a term in a document as a whole may be taken into account, when computing the influence of the term. Evidence also suggests that combining single terms into compound terms may be useful[34].

Bayesian network

All IR systems draw conclusions about the content of a document by examining some representation of that document. An automated system of indexing in such an approach bases its conclusions about

the document on the evidence of computable document features, such as the presence or absence of particular words and phrases [19].

A Bayesian network is a directed acyclic graph in which each node represents a random variable, that is a set of mutually exclusive and collectively exhaustive propositions. Each set of arcs into a node represents a probabilistic dependence between the node and its parents (the nodes at the other ends of the incoming arcs). A Bayesian network represents, through its structure, the conditional independence relations among the variables in the network. These independence relations provide a framework within which to acquire probabilistic information. A Bayesian network represents beliefs and knowledge about a particular class of situations. Given a Bayesian network for a class of situations and evidence about a particular situation in that class, conclusions about the document and document's relevant topics can be drawn [19].

The Bayesian network retrieval task is divided into three major steps:

1. Build the network representing the query.
2. Store each document
   (a) Extract the features from the document.
   (b) Instantiate the features in the retrieval network.
   (c) Calculate the posterior probability of relevance.
3. Rank documents according to the posteriors.

The advantages that Bayesian networks bring to the IR task include an intuitive representation of uncertain relationships and a set of efficient inference algorithms. Robert Fung and Brendan Del Favero [19] have used a probabilistic IR architecture that assists users who have fixed information needs in routing large amounts of material. Towards these goals, they have developed and implemented a system that allows a user to specify the topics of interest (i.e., information need), the quantitative and qualitative relationships among the topics, the document features, such as the

presence or absence of particular words and phrases, and the quantitative relationships between these features and the topics. [19].

3.1.2 Developments in Automatic Text Retrieval

In conventional information retrieval, the stored records are normally identified by sets of keywords or phrases known as index terms. Requests for information are typically expressed by Boolean combinations of index terms, consisting of search terms interrelated by the Boolean operators *and, or,* and *not.* The retrieval system is then designed to select those stored items that are identified by the exact combination of search terms specified in the available queries. The terms characterising the stored texts may be assigned manually by trained personnel, or automatic indexing methods may be used to handle the term assignment.

Refinements have been introduced into the Boolean processing environment. They allowed the terms assigned to documents to carry term weights. When term weights were introduced, they were called the fuzzy-set retrieval model.

3.2 Algorithms used by the IR community

The IR community's main concern is how to select significant words and phrases from a document that best describe the document or set of documents [36, 15]. Automatic summarisation of full documents generates a condensed version of the document[7]. The condensed version serves as an executive summary, which contains indicative information of the document's content. Automatic summarisation of full documents ascertains the relative importance of the material and generates coherent output[7]. The IR community has tried to automatically find significant words in documents and understand the content or meaning of the document. Although attempts have been made to utilise natural language text condensation approaches [45], they generally require the selection of a narrow domain and the availability of domain knowledge. These shortcomings made it infeasible

for generic text condensation tasks. The following subsections discuss some of the main ideas that make up the core of our system.

3.2.1 Luhn's ideas

Luhn assumes that frequency data can be used in extracting words and sentences that represent a document [36]. He ranked the words in the decreasing frequency of occurrence. After plotting the graph of frequency related to rank, he found that the curve was similar to the hyperbolic. This is in accordance with Zipf's law which states that the product of the frequency of use of words and the rank is approximately constant. He then excludes the non-significant words and the very high frequency words. Luhn also used this method to devise a method for automatic abstracting. He went on to develop a numerical measure of significance for sentences based on the number of significant and non-significant words in each portion of the sentence. Sentences were ranked according to their numerical score and only the highest ones would be included in the abstract.

3.2.2 C.J.van Rijsbergen's attempt

The document's representation aimed by Rijsbergen [46] consisted simply of a list of class names, each name representing a class of words occurring in the total input text. A document was indexed by a name if one of its significant words occurred as a member of that class. Such system consists of 3 parts:

1. Removal of high frequency words
2. Suffix stripping
3. Detecting equivalent stems

If two words have the same underlying stem, then they probably refer to the same concept and they should be indexed as such. It is inevitable that a processing system such as this will produce errors. Fortunately, experiments have shown that the error rate tends to be of the order of 5 per

cent [2]. Lovins [35] using a slightly different approach to stemming also quotes errors of the same order of magnitude. The final output would be a set of classes, one for each stem detected. A class name is assigned to a document if one of its members occurs as a significant word in the document.

3.2.3 Limitations of the Traditional Approaches

Traditional approaches to information retrieval use keyword searches and statistical techniques to retrieve relevant documents (e.g., [61, 53]). Statistical techniques take advantage of large document collections to automatically identify words that are useful indexing terms. However, word-based techniques have several limitations:

- **Synonymy:** Different words and phrases can express the same concept.
- **Polysemy:** Words can have multiple meanings [38].
- **Anaphora:** is a phenomenon of abbreviated subsequent reference to refer back to an entity introduced with more descriptive phrasing earlier by using a lexically and semantically abbreviated form [57]. It is used to make language more concise and avoid repetition and the most common manifestation of this is in the use of pronouns. For example in the following passage the anaphoric reference *their* refers to the earlier target *computers*:

  Computers are often mixed up with questions about *their* impact on ...

- **Phrases:** Some words are good indexing terms only in specific phrases.
- **Local Context:** Some words and phrases are good indexing terms only in specific local contexts.
- **Global Context:** Some documents do not contain any words or phrases that are good indexing terms.

3.2.4 Alternative Retrieval Models

The Vector Model

In the vector space model, documents are identified by sets of attributes, or terms. Instead of assuming that all terms are equally important, the system uses term weighting. The vector processing model offers simple, parallel treatments for both queries and documents. Extensions to the vector and Boolean models have been proposed including a generalised vector space model based on an orthogonal vector space. Another common retrieval model is the extended Boolean system which accommodates term weights assigned to both query and document terms as well as strictness indicators. The extended system thus covers vector processing, Boolean, and fuzzy set retrieval in a common framework, and it produces a vastly improved retrieval performance over simple Boolean operations.

The Probabilistic Model

The probabilistic retrieval model differs from those previously discussed. It represents an attempt to set the retrieval problem on a theoretical foundation. In the classical probabilistic models, the needed term probability is estimated by accumulating a number of user queries containing a term and determining the proportion of time a document is found relevant to the respective queries. Alternatively, a fixed query is considered and an attempt to determine the probability of an arbitrary document containing a query term will be judged relevant.

The probabilistic retrieval approach accommodates a large number of different phenomena about terms and documents as part of the probabilistic estimation process. This includes term co-occurence information, term relationships derived from dictionaries and thesauruses, and prior knowledge about the occurrence distributions of terms.

3.2.5 Enhancing the document representation

The conventional wisdom is that the keyword-type systems, where the information items are represented by sets of manually or automatically chosen index terms, have run their course. Most keywords are believed to be ambiguous and are often poorly represented by small collections of individual terms [54]. It is therefore widely believed that the keyword approach is not adequate for text content representation in information retrieval. By extension, the identification of text content by weighted term sets may also be unacceptable. Quoting from Blair: [6]

> No number of brute linguistic facts (word statistics) can be added up to give us the meaning of a text, where the meaning of a text would include such things as its subject, intellectual content, context, use, purpose, or links to other documents.

The available experimental evidence indicates that the use of abstracts in addition to titles brings substantial advantages in retrieval effectiveness. However, the additional utilisation of full texts of the documents appears to produce very little improvement over titles and abstracts alone in most subject areas [50]. This is one of the main reasons why the abstract is included in the Semantic Header.

3.3 Natural Language Processing in IR

3.3.1 Progress of Natural Language Processing in IR

The goal of an information retrieval system is to locate relevant documents in response to a user's query. Documents are typically retrieved as a ranked list, where the ranking is based on estimations of relevance [5]. Lexical ambiguity is a pervasive problem in natural language processing, and previous literature divides it into two types: syntactic and semantic [29].

To process a sentence of a language, the tokens must be isolated and identified. For Natural Language Processing, or NLP, lexical processing operates at the single word level and it involves identifying words and determining their grammatical classes or parts of speech so that higher levels

of language analysis can take place [57]. This usually consists of looking up a dictionary or lexicon, essentially a list of known words and their legitimate morphological variants. Ideally, lexical processing determines one base form for each word. Research into syntactic analysis of natural language has been concerned with the construction of wide coverage grammars[57].

The main sources of structural syntactic ambiguity in English are the attachment of prepositional phrases, the construction of nominal compounds and the scope of coordination and conjunction. For example, these two sentences:
Remove the bolt with the square head.
Remove the bolt with the square wrench.
are both lexically and syntactically identical but there is a genuine structural ambiguity as we do not know to what the prepositional phrases *with the square head* or *with the square wrench* should be attached. The case of nominal compounds occurs when a noun or nouns are used as modifiers of another noun, making a compound structure. Conjunction is one of the most frequently used constructions in natural language but the scope of the conjunts, i.e. what is being conjoined, can almost always be ambiguous.

The semantic level of language analysis is concerned with meaning and focuses on broad questions like what type of knowledge representation framework should be used [57]. On another level, there are semantic constraints on what should make semantically sensible natural language statements. The semantic level language analysis should be able to analyse grammatically parsed text into a knowledge representation. This is because a sentence may have a number of semantic interpretations, possibly arising from a number of syntactic interpretations, and as many of these should be eliminated.

The difficulty with semantic processing is that all the properties of every object and the legitimate arguments of all verbs must be known [57]. As a possible remedy to this problem, huge knowledge base could be built.

Detecting anaphora and resolving references would improve the understanding of a text. Even so, detecting anaphora is often difficult as there are no indicator phrases or terms. Some words are potentially anaphoric but not always so and anaphoric references can include many constructs. Although Liddy [13] lists almost 150 words which could be indicators of an anaphoric construct, the problem of reliably resolving anaphora still remains. It is important to note that fully-fledged NLP is being used in information retrieval [57]. This has led to the emergence of the application known as conceptual information retrieval. There, once the user requests information, he/she is given the information directly, instead of just receiving its reference.

Lexical level language processing in information retrieval

The simplest applications of NLP to information retrieval have been at the word level. Indexing based on some normalised or derived form of individual words occurs in the input [57]. An alternative to the popular stemming and conflation procedures would involve determining the base forms of words from a lexicon lookup. Building such a lexicon is expensive considering its marginal improvements over mechanical stemming. For those reasons, the idea has never really been pursued. However, lexical level language analysis has had a surge of interest recently with the increased availability of machine-readable dictionaries (MRDs)[57]. Its obvious use is to index by word senses rather than by word base forms. In information retrieval experiments, indexing by word senses using MRDs initially gave disappointing results in terms of retrieval effectiveness [57]. Because of this, researchers believe that it may not be necessary to determine the single correct sense of a word. Instead a sufficient understanding that allows one to rule out unlikely senses and to weigh likely senses highly, Krovetz and Croft stressed the importance of the word senses, that will provide a significant separation between relevant and non-relevant documents [29]. They mentioned that word ambiguity and the use of related or synonym words are two problems that arise when using words to represent the content of a document.

Syntactic level language processing in information retrieval

NLP techniques have been used to help index texts by elements more complex than word forms. Syntactic analysis can be used to analyse text in order to determine the boundaries of noun phrases which could then be used as internal representations. Indexing texts on a noun phrase basis using NLP techniques was done in the IOTA system [8]. One major problem of indexing by noun phrase units is the variety of ways of representing a complex concept in natural language. Three approaches have addressed the issue of ambiguity in syntactic analysis of texts for indexing purposes: ignoring ambiguity, normalising the identified phrases or indexing by structures which incorporate the ambiguities.

Ignoring the ambiguity allows texts to be indexed by phrases taken directly from the text. A large amount of work in this area has been done by Salton and others at Cornell University [55].

Normalising indexing phrases from texts and from queries into some standard form is used in the CLARIT project at Carnegie Mellon university [18]. A first order thesaurus for a domain, essentially a phrase list, is first generated automatically. Input texts are parsed and candidate noun phrases are identified. These are then compared to the thesaurus. They are classified as either:

1. exact (identical to some phrase in the list),
2. general (terms are constituents of those in the list), or
3. novel (new terms not on the list).

This approach always uses terms from the list as the indexing units and thus always yields the same syntactic form for a concept which could have been expressed in a number of different ways [18].

Encoding the ambiguity in some structure and allowing the retrieval operation to make allowances for this, handles the syntactic ambiguity in syntactically based indexing.

Semantic level language processing in information retrieval

Any piece of text which contains information essentially consists of a description of objects and actions on those objects. A number of conceptual information retrieval systems are described in the literature: SCISOR [23], RESEARCHER[32] and OpEd[1]. SCISOR [23] parses and analyses input stories into a knowledge base and then it answers users' questions about the content. RESEARCHER operates in the domain of US patent applications. Trying to resolve outstanding ambiguities, RESEARCHER uses limited semantics to resolve syntactic ambiguity and then uses the knowledge assimilated from the whole of the patent application it is processing [32]. OpEd is an editorial comprehension and question answering system which answers questions about beliefs, belief relationships and goals of those who have made arguments in the input texts [1].

3.4 Automatic Sentence Extraction used in Title and Abstract selection

Text processing methods based on a determination of term or sentence importance have been used not only for indexing but also for automatic abstracting purposes [52]. It was hypothesised that an extract of a document, that is a selection of significant sentences can serve as an abstract. This hypothesis concerning the substitutability of extracts for abstracts has been discussed in [15]. To achieve this, each sentence in the source text is scored according to some measure of importance, and the best rated sentences are selected [59]. Ideally, given a document represented as natural language text, one would like to construct a coherent well written abstract that informs the readers of the contents of the original, or at least indicates whether the full version may be of interest to the reader. A useful first step in the automatic or semi-automatic generation of abstracts from source texts is the selection of a small number of sentences, which are deemed to be important for purposes of content representation, from the source text [59].

The extraction methods used over the years start with a calculation of word and sentence significance, similar in spirit to the computation of the term weights. Criteria for the selection of important terms may be positional (the term's location in the document), semantic, or pragmatic (a system which would consider proper names as highly significant). Statistical term weights may be also criteria in selecting important terms. Since the frequency criteria are not very reliable, additional criteria should be used such as contextual inference (the word location or the presence of cue words), and syntactic coherence criteria [36, 16, 15, 49, 14, 4, 42, 43].

Kupiec et al.[30] describe a classification task on the basis of a corpus of technical papers with summaries written by professional abstractors. Their system identifies sentences in the text which also occur in the summary. Then it acquires a model of the abstract-worthiness of a sentence as a combination of a limited number of properties of that sentence. These properties include the sentence location in the document, the sentence length and the presence of thematic words in the sentence.

Simone H. Teufel and Marc Moens [59] report on a replication of Kupiec's experiment with different data. Summaries for their documents were not written by professional abstractors, but by the authors themselves. This produced fewer alignable sentences to train on. They used alternative meaningful sentences (selected by a human judge) as training and evaluation material, because this has advantages for the subsequent automatic generation of more flexible abstracts. They employed five different heuristics: four of the methods used by Kupeic et al as well as the title method described below. Kupeic et al's methods were the cue phrase method, location method, sentence length method and thematic word method.

1. *Cue phrase method*: it seeks to filter out meta-discourse from subject matter. Cue phrases were manually classified into five classes. This corresponds to the likely-hood of a sentence containing the given cue to be included in the summary. A score of minus one means very unlikely to be included in the summary, whereas a score of plus three means very likely to be

included in the summary.

2. *Location method*: Paragraphs at the start and at the end are more likely to contain material that is useful for a summary. These paragraphs tend to include crucial information. Simone et al's algorithm assigns non-zero values only to sentences which are in document peripheral sections. Sentences in the middle receive a zero score.

3. *Sentence length method*: All sentences under a certain length (fifteen tokens including punctuation are given a score of zero. All sentences above that criterion are assigned a score of one.

4. *Thematic word method*: It identifies key words that are characteristic for the contents of the document. The top ten scoring words are chosen as the thematic words. Sentence scores are then computed as a weighted count of thematic words in that sentence.

5. *Title method*: Words occurring in the title are good candidates for document specific concepts. Simone et al. also experimented by taking into accounts words occurring in the headings. Better results were generated using title words only [59].

3.5 Text Classification/Categorisation

An important step in building up the document database of a full text retrieval system is to classify each document under one or more classes according to the topical domains that the document discusses. This is commonly referred to as classification. Automatic classification has two major components:[24]

1. The classification scheme, which defines the available classes under which a document can be classified and their inter-relationships.

2. The classification algorithm, which defines the rules and procedures for assigning a document to one or more classes.

Wong, Kan and Young presented an automatic classification approach called ACTION [24]. The key idea behind it is a scheme for measuring the significance of each keyword in a given document. That scheme takes into account not only the occurrence frequency of a keyword, but also the logical relationship between the available classes.

Text categorisation systems assign predefined category labels to texts. For example, a text categorisation system for computer science might use categories such as operating systems, programming languages, AI or information retrieval [47]. Text Categorisation are typically applied to static databases [47].

The relevancy signatures algorithm [48] uses linguistic phrases, the augmented relevancy signatures algorithm uses phrases and local context, and the case-based text classification algorithm uses larger pieces of context. These three algorithms were evaluated and the results suggested that information extraction techniques can support high-precision text classification. In general, using more extracted information improves performance.

There have been approaches using knowledge bases relying on a domain-specific dictionary to drive the information extraction system [48]. It seems reasonable to believe that we could produce accurate classifications if we could actually understand the documents. However, natural language understanding is an expensive endeavour that can strain computational resources. Thus, some researches have turned their attention to information extraction that extracts specific types of information from a document [48]. For example, in the domain of terrorism, an information extraction system might extract the names of all perpetrators, victims and weapons that were involved in a terrorist attack. The main advantage of this approach is that portions of a text that are not relevant to the domain can be effectively ignored. Since the system is only concerned with the domain-specific portions of the text, some of the most difficult problems in NLP are simplified. As a result, information extraction is a practical and feasible technology that has achieved success in the last few years [33, 48].

Edmundson [17] describes new methods of automatically extracting sentences from documents for screening purposes. His method describes the sentence significance, the high content words previously described and three additional components: pragmatic words (cue words), words found in the title and the headings, and the structural indicators (sentence location). An attempt was made by Edmundson to classify eligible sentences as to qualitative degree of extract-worthiness. In practice, however, it did not prove satisfactory for sentence selection. The principles he followed in devising the guide to the development of automatic extracting methods so as to yield close approximations to target extracts were:

1. Detect and use all content and format clues to the relative importance of sentences that were originally provided by the author, editor or printer.
2. Employ a system of reward weights for desired sentences and penalty for undesired sentences.
3. Employ a system of parameters that can be varied to permit different specifications for extracts.
4. Employ a method that is a function of several linguistic factors (syntactic, semantic, statistical locational, etc.).

Thus, the four basic methods Edmundson used in his automatic extracting system are the Cue, Key, Title and location methods.

Clearly, there are extracting clues that have not been exploited-in captions of figures and tables, in footnotes and references.

Automatic News Extraction System

The Automatic News Extraction System (ANES)[7] initial technical approach was similar to that taken by Johnson et al[26]. In addition to constructing summaries by extracting whole sentences from the source documents, Johnson also used such phrases as "*the objective of this study...*" to indicate key content. The ANES process of summary generation is made up of four major constituents: statistical corpus analysis, signature word selection by applying a term frequency and the inverted

document frequency model, sentence weighting, which is computed by summing the weights of the individual signature words present in the sentence, and sentence selection[7].

3.6 Retrieval and Information extraction systems

3.6.1 The SMART Retrieval System (Salton)

The SMART system is a sophisticated text retrieval tool based on storing all information terms in a vector of terms. In principle, the terms might be chosen from a controlled vocabulary list or a thesaurus [55]. After performing the SMART experiments [51], some conclusions were drawn. The main conclusions are listed below:

Document length Document abstracts are more effective for automatic content analysis purposes than are document titles. Improvements appear possible if the abstracts were replaced by larger text portions. The improvement, however, is not large enough to conclude that full text processing is superior to abstract processing.

Synonym Recognition Dictionaries providing synonym recognition produce statistically significant improvements in retrieval effectiveness compared with word stem matching procedures.

Order of merit A summary of the results after applying the SMART system shows that abstract processing with phrase and synonym recognition had the best results. Next most effective were the results that were drawn from using weighted word stem matching and statistical word associations using abstracts for analysis purposes. The less effective results were upon matching logical word stem and disregarding term weights. The least effective results were when only document titles were used for analysis purposes.

3.6.2 Oracle ConText-Text Management System

One of the most critical challenges facing business today is managing information. Unstructured, primarily textual information, becomes trapped as essentially dead fields in traditional databases. As a result, information that resides as text documents, manuals, reports, e-mail and faxes has been largely inaccessible to the corporate decision makers who need it most [66, 67].

The Oracle7 ConText option is a fully integrated text management solution that enables users to process text-based information as quickly and easily as relational data. Oracle context analyses the contents and understands the structure of the English text it reads. The Oracle Release 7.3 ConText option consists of two separate, yet closely interrelated functions: a text management architecture contained entirely within Oracle7 and a text retrieval feature which uses natural language processing technology to identify themes and content in text. It is also capable of analysing the thematic content stored text and generating automatic summaries. ConText's core is its parser, a sophisticated and robust collection of lexical attributes (dictionary information) and parsing rules. The parser consists of five data collection layers. These layers are :

1. Syntax, theme and grammar layers which analyse documents at the sentence level and produce a structural and thematic representation of sentences.

2. Connotation and discourse layers which analyse documents at greater-than-sentence level and produce a representation of discourse dynamics.

In other words, these modules track the rise and decay of themes/subjects, and assign a subject matter description of sentences, paragraphs, and documents. In addition, ConText's Concept Processing Engine represents an impressive English language knowledge base.

Systems that rely on simple word recognition and repetition often miss the actual meaning of a document and as a result produce an enormous amount of irrelevant output. By breaking down the text into its constituent grammatical elements and determining how these elements contribute to

the overall meaning, ConText works to understand the text it processes. It then uses this knowledge to produce a database index which can identify the development of key themes and determines their relative prominence [40]. Unlike other products that simply count words or use a hierarchical thesaurus to determine the main theme of a document, ConText parses every sentence in a document to determine the relative weight of the different themes.

The Oracle ConText Lexicon is the heart of this text retrieval system. The Oracle ConText Lexicon contains a vast dictionary of over 1,000,000 words and phrases as well as the linguistic rules that bind them into thematic units. The lexicon is designed to recognise the vocabulary used in over 1,000 industries and can be augmented by user dictionaries.

The ConText option provides automatic text reduction, which creates summaries conveying the main ideas and concepts of full documents. Text reduction condenses large documents to a manageable level. In addition to text reduction, ConText contains a powerful text classification feature which categorises documents according to linguistically identified themes rather than word frequency and statistics.

3.6.3 Nordic WAIS/World Wide Web Project

The Nordic WAIS/World Wide Web Project works on Improving Resource Discovery and Retrieval on the Internet [3]. The project has three main parts:

1. A demonstration of how an existing library system can be integrated in the world of open client/server networking.

2. Simplifying the use of Wide Area Information Server, or WAIS, database servers by establishing a World Wide Web, or WWW, front-end service. Two activities are involved:

   (a) Constructing a subject tree of WAIS databases which can be browsed within WWW.

   (b) Enhancing the current WWW gateway to WAIS to support the full WAIS functionality.

3. Simplifying the use of WWW by supplementing the hypertext browsing with a search option, based on a WAIS index of WWW resources.

In using WWW as a smart front-end to WAIS databases, a set of cooperating programs that implement automatic indexing and classification of WAIS databases has been developed by the Nordic project. The Nordic's automatic classification depends on UDC [3], an English medium classification scheme. The dynamic nature of the information sources on the network makes it necessary to have automatic tools that index and classify material. The algorithm used is as follows:

1. From the different fields of the selected document, words are extracted into a number of groups:
   - words from the description field
   - words from the subject field
   - words from the keyword-list field
   - words from the description field marked as keywords together with the name of the database.
2. A list of suggested classifications is constructed by comparing words from these groups with UDC's vocabulary. When a match between the vocabulary and a word is found the corresponding classification is added (restricted to the top 2 levels) to the list of suggested classifications with a certain weight. The weight depends on which group the matching word originates from. As an illustration, keywords in the subject field have higher weights than ordinary words in the description field.
3. From the list of suggested classifications, the final classification is made. It is based on the accumulated weights for each proposed classification.

The Nordic project is not tied to UDC but can be used with other classification schemes such as the Library of Congress in order to produce different views of the subject trees.

3.6.4 Harvest's Essence

Overview of Harvest's Essence

Essence's main objective is to extract indexing information from an input document. Content indexing requires type-specific processing to extract information effectively. By exploiting the semantics of common file types[1], Essence generates compact yet representative file summaries that can be used to improve both browsing and indexing in resource discovery systems [22].

Essence decomposition

Essence decomposes the information extraction problem into four components that are independent of how data are stored, updated or exported. The components are listed below:

1. The type recognition step that uses various methods to determine a file's type
2. The presentation unnesting step that transforms nested files into an unnested format.
3. Candidate selection step, selects which objects are to be summarised.
4. The summarising step, which applies a type specific extraction procedure to each selected object.

Type Recognition

Essence recognises file types using a combination of file and site naming conventions, content testing, and user defined methods. It can also use explicit file typing information for environments that contain such information. The two main type recognition steps are:

- Naming conventions and heuristics.
- Examining file contents in determining the file types.

[1] See The Summariser's functions for each document table at the end of this subsection

Presentation Unnesting

The type recognition step sometimes encounters files encoded to one or more presentation layer data transformations. These transformations arise because of heterogeneity and other complexities in a distributed environment. They include operations such as compression and ASCII encoding.

When a presentation nested file is encountered, it is unnested into one or more result files. The result files themselves can also be nested. In addition to unnesting the input files, the presentation unnesting step also keeps a record of the nested origin of each unnested file, for use by the candidate selection and summarising steps.

Candidate Selection

Given a set of typed objects, the candidate selection step chooses objects to summarise. It attempts to eliminate redundancy among related files.

Essence Summarising Subsystems

The summarising step applies an extraction procedure, called a summariser, to each selected object, based on the type of information uncovered in the type recognition step. Each summariser is associated with a specific file type; it understands the type well enough to extract summary information from the file.

Type	Summariser Function
Audio	Extract file name
Bibliographic	Extract author and titles
Binary	Extract meaningful strings and manual page summary
C, CHeader	Extract procedure names, included file names, and comments
Dvi	Invoke the Text summariser on extracted ASCII text
FAQ, FullText, README	Extract all words in file
Framemaker	Up-convert to SGML and pass through SGML summariser
Font	Extract comments
HTML	Extract anchors, hypertext links, and selected fields
LaTex	Parse selected LaTex fields (author, title, etc.)
Mail	Extract certain header fields
Makefile	Extract comments and target names
ManPage	Extract synopsis, author, title, etc., based on "-man" macros
News	Extract certain header fields
Object	Extract symbol table
Patch	Extract patched file names
Perl	Extract procedure names and comments
PostScript	Extract text in word processorspecific fashion, and pass through Text
	summariser.
RCS, SCCS	Extract revision control summary
RTF	Up-convert to SGML and pass through SGML summariser
SGML	Extract fields named in extraction table
ShellScript	Extract comments
SourceDistribution	Extract full text of README file and comments from Makefile
	and source code files, and summarise any manual pages
SymbolicLink	Extract file name, owner, and date created
Tex	Invoke the Text summariser on extracted ASCII text
Text	Extract first 100 lines plus first sentence of each remaining paragraph
Troff	Extract author, title, etc., based on "-man", "-ms", "-me" macro
	packages, or extract section headers and topic sentences.
Unrecognised	Extract file name, owner, and date created.

Table 1: The Harvest's Summarisers' functions for each document

3.7 Conclusions

In this chapter, we described approaches in information retrieval, document indexing and text classification. We will be using some of these ideas in our system. Since the term position in the document is weighted, we will give an importance of the term location in the document. Since some of the available experimental evidence indicates that the use of abstracts in addition to titles brings substantial advantages in retrieval effectiveness [50], and since Salton's SMART system reveals that using the abstract rather than the whole text gives the best results in information retrieval, the abstract and the title are used as two of the components of the Semantic Header. Since the titles could be good candidates for document specific concepts as Simone et al stressed, we will assign high weights to the terms located in the abstract and title fields. The additional utilisation of full texts of the documents appears to produce very little improvement over titles and abstracts alone in most subject areas [50]. In addition to assigning term weights, our system used the term frequency of occurrence addressed by Luhn.

The automatic classification approach used in ACTION relates the significant keywords to a set of available classes. Our system's thesaurus concept will be based on this idea; however, our system relates controlled terms with a set of subject headings. Our document classification scheme is based on Nordic's classification scheme. Nordic classifies documents by looking for a match between a set of vocabulary and the words in the document. Nordic uses words extracted from a set of groups and UDC's vocabulary to classify a document. Each classification gets a weight depending on which group the matching words originated from. The classification having the highest weight is selected. Our system will look for a match between a set of different weighted terms generated from the document and a set of controlled terms. The highest weighted subject headings associated with the matched controlled terms will be selected.

Luhn's automatic abstracting idea will be used in generating an abstract for a document and Harvest's file type recognition will be implemented in our thesis.

Chapter 4

ASHG's Thesaurus

4.1 The Thesaurus in IR

A thesaurus is a set of items (phrases or words) plus a set of relations between these items [25]. The Thesauri commonly used in IR have shown inconsistent effects on retrieval effectiveness, and there is a lack of viable approaches for building a thesaurus automatically [25].

There are two types of manual thesauri. The general purpose and word based thesauri like Roget's and WordNet contain sense relations such as antonym and synonym but are rarely used in IR systems. The IR oriented and phrase based thesauri such as INSPEC, Library of Congress Subject Headings (LCSH), and Medical Subject Headings (MeSH) are widely used in commercial systems [25]. The major drawback of both types are that they are expensive to build and hard to update in a timely manner.

This thesis addresses the issue of constructing a thesaurus in a semi-automatic fashion. We used a number of rules in merging the subject headings found in INSPEC [65], LCSH [69] and ACM [70].

4.2 The Thesaurus for ASHG

The ASHG's Thesaurus is composed of a three level subject hierarchies and a set of control terms associated with the subject headings found in the subject hierarchies. The Thesaurus used by ASHG contains four object classes: *Level_0* which represents the general subject of the subject hierarchy, *Level_1* which represents the sub-subject of the general subject and is derived from *Level_0*; *Level_2* which represents the sub-subject of the *Level_1* subject and is derived from *Level_1*, and finally *Control_term* which contains the root terms that are derived from the subject headings. A root term is the origin of all possible terms that can be generated from it by adding the suffixes and prefixes.

4.2.1 The Subject Hierarchies

Since different subject headings may be used to convey the same subject, and since different people may have different perspectives on the same single subject, controlled subject headings were derived. The CINDI system focuses on the standardisation of subject headings. This database helps the provider of the primary resource in selecting the correct subjects and sub-subjects' headings for the semantic header entry.

CINDI's subject hierarchy is made up of three levels, where *level_0* contains the general subject heading. Currently we have included only two general subject headings: Computer Science and Electrical Engineering. *Level_1* contains all the subjects that fall under *level_0* subjects, and similarly *level_2* will contain more precise subjects that fall under *level_1* subjects.

4.2.2 Building CINDI's Classification

ACM, INSPEC and LCSH were the main building blocks of CINDI's three level Subject Hierarchy. ASHG's computer science subject hierarchy used ACM's subject hierarchy as the starting point, and ASHG's electrical engineering subject hierarchy was based on INSPEC's subject hierarchy. We have exploited LCSH's subject headings relations to refine both hierarchies. LCSH contained relations between subject headings such as BT (Broader Term), NT (Narrow Term), UF (Used For), and RT

(Related To). In order to augment ACM and INSPEC subject hierarchies, a search for an ACM or INSPEC subject heading was made in LCSH. If a match was found, the narrow terms found in LCSH under the matched subject were added to the list of subjects or terms under the ACM or INSPEC's matched subject heading.

This augmentation produced a hierarchy composed of five or six levels. Since CINDI's subject hierarchy was limited to only three levels, rules were applied to merge these subject headings. The resulting subject hierarchy was formed of three level subject hierarchy and one additional level. This last level contains terms used as control terms associated with the *Level_2* subject headings.

Rules used in Merging the subjects of different levels

1. The Computer Science's subject hierarchy's general (*Level_0*) subject is *Computer Science*. The Electrical Engineering's subject hierarchy's general (*Level_0*) subject is *Electrical Engineering*. Similarly, other subject hierarchies will be disciplined based as in the LCSH.

2. *Level_1* and *Level_2* subject headings found in the augmented ACM (or INSPEC) were merged to form one level, the CINDI's *Level_1* subject heading. For some of the subject headings found in *Level_2* which contained the subject headings found in *Level_1*, the *Level_1* subject headings were dropped. The same rule was applied on subject headings found in *Level_3* and *Level_4* to give the CINDI's *Level_2* subject heading. For example, *Software* and *Software Engineering* were found in the augmented ACM *Level_1* and *Level_2* subject headings respectively. We dropped *Software*, to yield *Software Engineering* as CINDI's *Level_1* subject heading.

3. Some of the subject headings found in the *Level_1* and *Level_2* augmented ACM (or INSPEC) subject hierarchies were concatenated with a colon to form the CINDI's *Level_1* subject heading. The same rule was applied on subject headings found in the augmented ACM (or INSPEC) *Level_3* and *Level_4* to yield CINDI's *Level_2* subject heading. For instance, *Office Automation* and *Spreadsheets* were found in the augmented ACM (or INSPEC) *Level_3* and *Level_4* subject headings respectively. We concatenated them to derive CINDI's *Level_2* subject heading,

Office Automation: Spreadsheets.

4. The *Level_5* and *Level_6* augmented ACM (or INSPEC's) subjects were used as controlled terms associated with CINDI's *Level_2* subject headings. *Copyrights*, for example, was used as a control term associated with CINDI's *Level_2* subject heading, *Hardware and Software Protection*

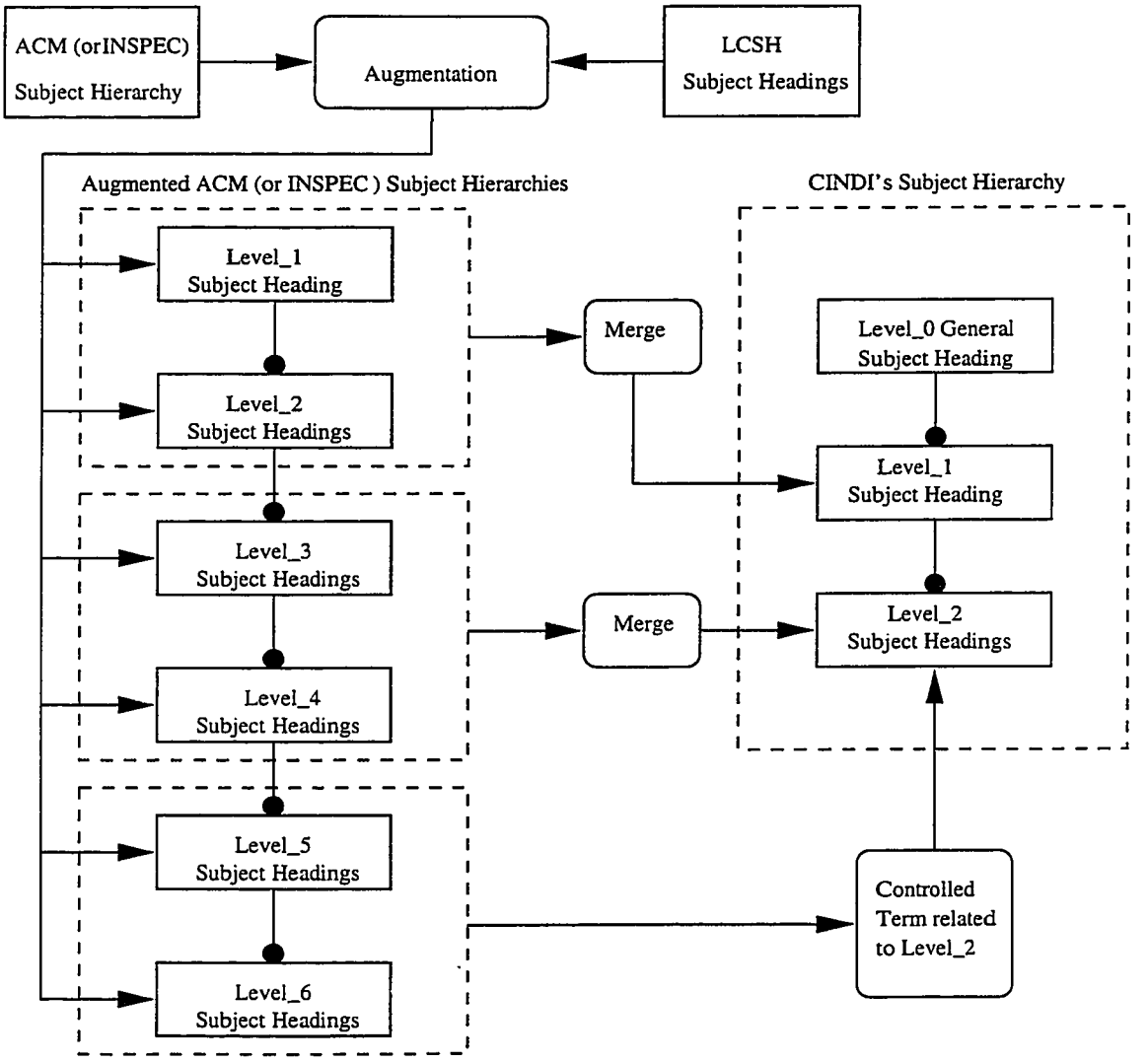

Figure 4: Transforming ACM (or INSPEC) Subject Hierarchy into CINDI's Subject Hierarchy

The subject headings' classes

Having described the building of CINDI's three level subject hierarchy from an augmented ACM and INSPEC subject hierarchies, we present the three level subject headings classes:

```
persistent class Level_0  {
  protected:
    indexable char lev_0[MAX1];
  public:
    Level_0(char * lv_0);
    char* get_lev_0() { return lev_0; };
    virtual char* get_subject() { return lev_0; };
    virtual void print();
    persistent Level_0* isLevel_0(char *str);
    void list_all_level_0();
    int list_substr_0(char* str, int count);
};

persistent class Level_1  :  public  Level_0  {
  protected:
    indexable char lev_1[MAX1];
  public:
    Level_1 (char *lv_1, char *lv_0);
    virtual void print();
    virtual void print_all();
    persistent Level_1* isLevel_1(char *str);
    void list_all_level_1(char* lev_0);
    int list_substr_1(char* str, int count);
    char *get_lev_1() { return lev_1; }
    virtual char* get_subject() { return lev_1; };
    char* get_subject_0(char *str);
};

persistent class Level_2  :  public  Level_1  {
  protected:
    indexable char lev_2[MAX1];
```

```
  public:
    Level_2 (char *lv_2, char *lv_1, char *lv_0 );
    virtual void print();
    virtual void print_all();
    persistent Level_2* isLevel_2(char *str);
    void list_all_level_2(char* lev_0, char* lev_1);
    int list_substr_2(char* str, int count);
    char* get_lev_2() { return lev_2; }
    virtual char* get_subject() { return lev_2; };
    char* get_subject_1(char *str);
};
```

Since the three subject headings classes have similar characteristics, we will only describe the methods of the *Level_2* class. The constructor *Level_2* initialises *Level_2*'s and its parents' values. The method *print()* prints *Level_2* object into the result file. The method *print_all()* prints *Level_2* object as well as its parent objects *Level_1* and *Level_0*. The method *isLevel_2* returns a *Level_2* object whose value is 'str', otherwise it returns 0 or NULL pointer. The method *list_all_level_2* prints into the result file all *Level_2* objects where their *Level_0* and *Level_1* subject values correspond to 'lev_0' and 'lev_1', respectively. Its definition follows:

```
persistent Level_2 *Lev_2;
readonly  trans  {
    for (Lev_2 in Level_2) suchthat (strcmp(Lev_2->lev_0, lev_0) == 0
                                    && strcmp(Lev_2->lev_1, lev_1) == 0)
        Lev_2->print();
}
```

The method *list_substr_2* prints all objects each of whose value starts with the string 'str', and it returns the number of such objects found. The definition of this method is:

```
persistent Level_2 *Lev_2;
readonly  trans  {
    for (Lev_2 in Level_2) suchthat (starts_with(Lev_2->lev_2,str) == 1) {
Lev_2->print_all();
++count;
    }
}
```

The methods *get_lev_2* and *get_subject* return a pointer to the *Level_2* subject of the current object. Both are used in document classification. The method *get_subject_1* returns a pointer to the *Level_1* object which is the parent of a *Level_2* object whose value is the string 'str'. Its definition follows:

```
persistent Level_2 *Lev_2;
for (Lev_2 in Level_2) suchthat (cmp_case(Lev_2->get_lev_2(),str) == 0)
   return Lev_2->lev_1;
```

4.2.3 The Control Term Subject Association

The CINDI system uses a thesaurus to help the user in the registering and search processes. One such need for a thesaurus is in avoiding chaos introduced by differences in perception of different indexer. Hence, some form of standardisation of terms used has to be enforced.

The main reason behind the Control Term Subject association is to extract or classify the primary source under a number of subject headings by comparing the significant list of words contained in the document with the list of controlled terms. An association between the controlled terms and their corresponding subject headings is created.

Each controlled term has three lists of subject headings attached to it. The three lists correspond to the general subject headings, sub-subject *Level_1* subject headings, and *Level_2* subject headings. Our controlled terms were based on the terms found in CINDI's subject hierarchy and the additional terms that are associated with CINDI's *Level_2* subject headings. For each subject heading found in CINDI's subject hierarchy and the additional terms, we used their constituent English none noise words as their corresponding controlled terms. For example, the control term *compute* will be associated with *Computer Science* general subject heading. Similarly, the control term *hardware* will be associated with *Hardware integrated circuits* and *Hardware performance and reliability* level_1 subject headings and *Hardware Simulation Design Aids* level_2 subject heading. Each controlled term is associated with one or more subject headings.

4.2.4 Building the Controlled Terms

The subject headings found in CINDI's *Level_0*, *Level_1* and *Level_2* will be used as the basis for finding the controlled terms. In addition, the additional terms associated with CINDI's *Level_2* subject headings are mapped into some controlled terms.

Deriving the controlled terms

Mapping CINDI's subject headings terms into controlled terms involves:

1. Since the controlled term dictionary is only composed of significant words, English stop words are removed from CINDI's subject hierarchy headings and the additional terms associated with CINDI's *Level_2* subject headings. English Stop words are found in Table 2.

2. Applying ASHG's stemming process to the remaining list of words in order to get their root, which will be stored in the list of controlled terms.

3. Generating a list of words to be added to the *spell* check dictionary. These words are found in the subject headings but not in the *spell* check dictionary. Words like *WWW* would be checked as wrong by the Unix *spell* check command, because *WWW* is not found in the *spell*'s dictionary. So, *WWW* should be an added to the list of controlled terms.

Associating the controlled terms with the subject headings

A document often covers a number of subjects or domains. Naturally some of them are of higher importance than others. CINDI uses the words in a document to classify it under a list of subject headings. This list of words from the document are matched against the controlled terms, generated above. The association between the subject headings and the controlled terms is constructed by comparing the root words found in these subject headings with the CINDI's controlled terms. If a match is found, then this subject heading is associated with the controlled term. The reason behind building such an association is that ASHG will generate a suggested list of subject headings using

the words found in the document by consulting the Controlled term subject association. A summary of the steps used is discussed below:

1. Split each subject heading and the terms associated with CINDI's *Level_2* subject headings into the words they are made up of.
2. English noise words found in the list of words are removed.
3. Words are checked using the *spell* command.
4. Similarly, words not found in the *spell* Unix dictionary and the new added words are dropped.
5. Apply the stemming process to generate the root controlled terms from the words.
6. Each root controlled term will be associated with the subject headings that contains it.

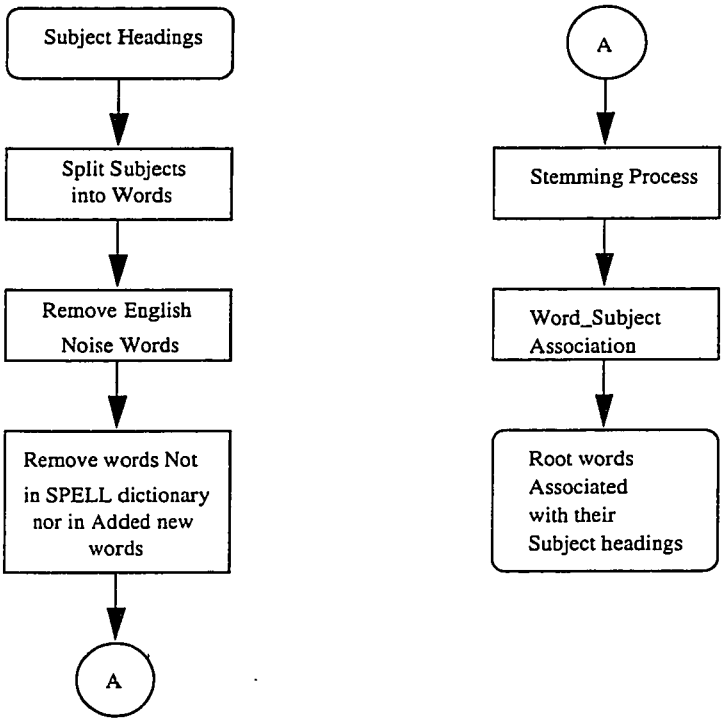

Figure 5: Associating words' roots to their subject headings

For example, *Theory of computation by abstract devices* subject heading is divided into the following words: *theory, of, computation, abstract, by,* and *devices.* The English noise words such

as *of* and *by* are dropped. Steps 3, 4 and 5 are applied on the remaining terms. The generated root terms such as *abstract, theory, computation*, and *device* are associated with the initial subject heading.

The Control Term's class

Having described the controlled term subject association, we present the control term class:

```
template<class T>
persistent class _SubjectPtrWrapper {
public:
    persistent T * info;
    _SubjectPtrWrapper(persistent T *inf) : info(inf) { }
    _SubjectPtrWrapper() : info(NULL) { }
    _SubjectPtrWrapper(const _SubjectPtrWrapper<T>& wrap) { info=wrap.info; }
    int match (const _SubjectPtrWrapper<T>& wrap) { return (info==wrap.info); }
    void print() const { info ->print(); }
    char* Getsubject() const {return info->get_subject(); }
};

/* The ODE's template feature for _SubjectPtrWrapper<T> did not
 * work out so this redundant type definitions were used.
 */

typedef _SubjectPtrWrapper<Level_0> Level_0_wrap;
typedef _SubjectPtrWrapper<Level_1> Level_1_wrap;
typedef _SubjectPtrWrapper<Level_2> Level_2_wrap;

persistent class stack_array_0 {
  public:
    stack_array_0(int = 1);              /* Constructor */
    void push(persistent Level_0 *);     /* Adds a persistent level_0 subject */
    int isempty() const {return num == -1; }
    int isfull();                        /* Check if the array is full */
    persistent Level_0_wrap * get_ptr() { return Stackptr; }
```

```
    int get_num() {return num;}
    int get_size() { return size; }
    int search(persistent Level_0 *);   /* Search for level_0  in array */
    void remove(persistent Level_0 *);  /* Removes the pointer to level_0
 * from array
 */
    void print_all();
 private:
    int size;
    int num;
    persistent Level_0_wrap * Stackptr; /* Pointer to the array of level_0
 * wrapper.
 */
};
```

There are three similar classes for *stack_array* for the three subject headings levels. Therefore, we will describe the *stack_array_0* class. It contains three private data members: *size*, which is the size of the array, *num*, which is the number of elements in the array, and *Stackptr*, which is a pointer to a persistent *Level_0_wrap* type. *Level_0_wrap* is *_SubjectPtrWrapper<Level_0>*, which is a wrapper for subject level_0 headings.

The constructor *stack_array_0* initializes the array by creating an array of *Level_0_wrap* of size *s* or one, and it sets the *size* and *num*. Its definition follows:

```
stack_array_0::stack_array_0(int s)
{
    size = s;
    num = 0;
    Stackptr = pnew Level_0_wrap[size];
}
```

The method *isempty* checks if array is empty. The methods *get_ptr*, *get_num*, and *get_size* are defined in the class description. The method *push* adds a pointer to a subject heading into the

array. If the array is full, then a new array of a bigger size is created, the old one is copied into the new one and the new pointer is then added to the array. Its definition is shown below:

```
void stack_array_0::push(persistent Level_0 * item)
{
    int i = isfull();
    if (i != -1) {
Stackptr[i].info = item;
num++;
    }
    else {
persistent Level_0_wrap * new_list = pnew Level_0_wrap[size+20];
for (i=0; i<size ; i++)
    if (Stackptr[i].info != NULL) {
new_list[i].info = Stackptr[i].info;
    }
pdelete [] Stackptr;
Stackptr = new_list;
Stackptr[size].info = item;
num++;
size += 20;
    }
}
```

The method *isfull* checks if the array is full. Its definition follows:

```
int stack_array_0::isfull()
{
    for (int i=0 ; i<size; i++)
        if (Stackptr[i].info == NULL)
           return i;
    return -1;
}
```

The method *search* looks for a pointer to a subject heading in the array. If the pointer to the subject heading *sub0* is found in the array, its position is returned, otherwise minus one is returned.

```
int stack_array_0::search(persistent Level_0 * sub0)
{
    for (int i=0; i<size; i++)
if (Stackptr[i].info != NULL)
    if (Stackptr[i].info == sub0)
return i;
    return -1;
}
```

The method *remove* removes a pointer to a subject heading from the array. It decrements the number of elements and sets the pointer to NULL. Its definition follows:

```
void stack_array_0::remove(persistent Level_0 * sub0)
{
    int i = search(sub0);
    if (i > -1) {
Stackptr[i].info = NULL;
num--;
    }
}
```

Method *print_all* prints all the subject headings pointed by in the array. Its definition follows:

```
void stack_array_0::print_all()
{
    for (int i=0; i<size; i++)
if (Stackptr[i].info != NULL)
    printf("%s\n",Stackptr[i].Getsubject());
}
```

The class Keyword interface follows:

```
persistent class Keyword {
    friend void trace_list_0(persistent Keyword *, SNODE *, int);
    friend void trace_list_1(persistent Keyword *, SNODE *, int);
    friend void trace_list_2(persistent Keyword *, SNODE *, int);
```

```
  private:
    indexable char Key[MAX1];
    persistent stack_array_0 * Lev_0_list;
    persistent stack_array_1 * Lev_1_list;
    persistent stack_array_2 * Lev_2_list;
  public:
    Keyword(char *s);
    void Push_0(persistent Level_0 * t) { Lev_0_list->push(t); }
    void Push_1(persistent Level_1 * t) { Lev_1_list->push(t); }
    void Push_2(persistent Level_2 * t) { Lev_2_list->push(t); }
    int Find_0(persistent Level_0 * t) { return Lev_0_list->search(t); }
    int Find_1(persistent Level_1 * t) { return Lev_1_list->search(t); }
    int Find_2(persistent Level_2 * t) { return Lev_2_list->search(t); }
    void Remove_0(persistent Level_0 * t) { Lev_0_list->remove(t); }
    void Remove_1(persistent Level_1 * t) { Lev_1_list->remove(t); }
    void Remove_2(persistent Level_2 * t) { Lev_2_list->remove(t); }
    void Print_0() { Lev_0_list->print_all(); }
    void Print_1() { Lev_1_list->print_all(); }
    void Print_2() { Lev_2_list->print_all(); }
    persistent stack_array_0 * Get_Lev_0() { return Lev_0_list; }
    persistent stack_array_1 * Get_Lev_1() { return Lev_1_list; }
    persistent stack_array_2 * Get_Lev_2() { return Lev_2_list; }
    virtual void print() { printf(" %s\n",Key); }
    char *Getkeyword() { return Key; }
    persistent Keyword* isKey(char *str);
};
```

NOTE: There is a duplication of the member functions, except that each function handled different subject level headings. That was due to an ODE error caused by assigning persistent and volatile pointers inside the body of these functions. A template for the *stack_array_0*, *stack_array_1* and *stack_array_2* can be created if this bug is fixed.

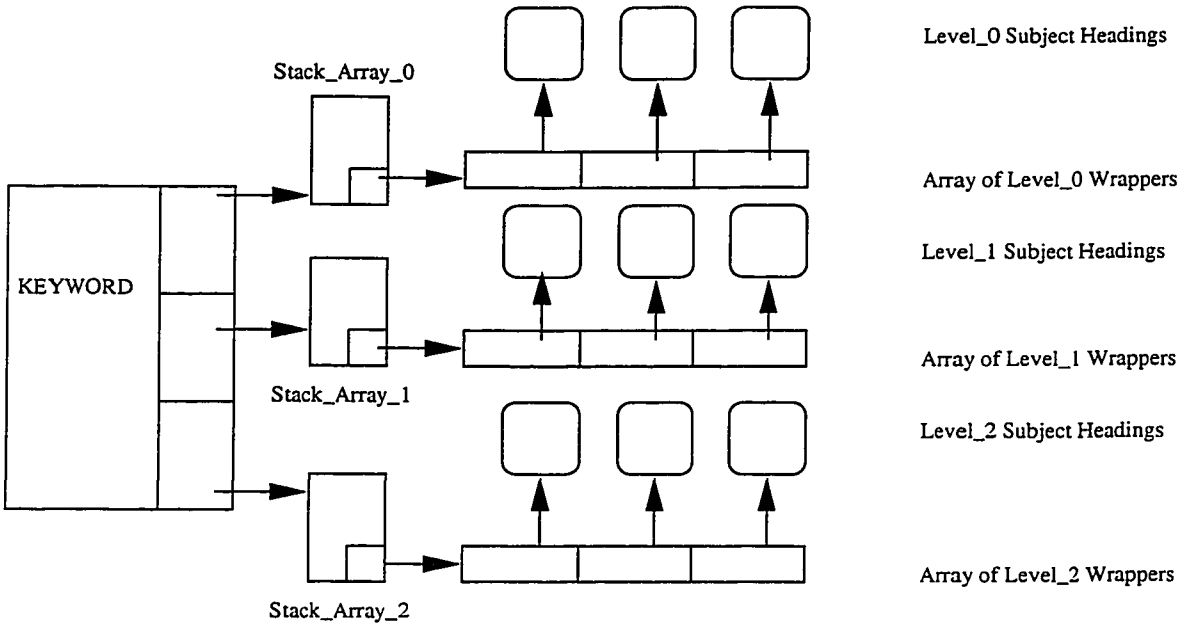

Figure 6: Keyword Class Definition

The class control term has three friend functions, that will be used in the subject extraction process. It also has four private data members: The *Key* which holds the value of the keyword, and three pointers pointing to *stack_array_0*, *stack_array_1* and *stack_array_2* respectively. It contains redundant functions as mentioned in the *NOTE* above. Its constructor *Keyword* copies *s* into the *Key* value, and creates three pointers. Its definition is as given below:

```
Keyword::Keyword(char *s)
{
    strcpy(Key,s);
    Lev_0_list = pnew stack_array_0;
    Lev_1_list = pnew stack_array_1;
    Lev_2_list = pnew stack_array_2;
}
```

The method *isKey* checks if the string *str* is a keyword. If it is it returns a pointer to it, otherwise, it returns zero. Its definition is as given below:

```
persistent Keyword* Keyword::isKey(char *str)
```

```
{
    persistent Keyword *k;

    for (k in Keyword) suchthat (cmp_case(k->Key,str) == 0) {
    return k;
    }
    return 0;
}
```

In the next section, we will describe how the thesaurus is automatically built.

4.2.5 Programs used to build the Thesaurus

Having described both procedures used in generating the subject headings hierarchy and the association between words' roots and their CINDI's subject headings, we move on to describe the programs that will store them in CINDI's Thesaurus.

new-subject-build.cc After merging the subject levels, the subject headings are passed to a function called *build_sub2*. This function will generate or distribute the *Level_1* subject to *Level_2* subject headings. The output of this function is a file having _sub2 extension. The executable file *new_subject_build.cc* is applied on the outputed file producing the three level subject headings hierarchy. Here is the algorithm used in this program:

1. Open CINDI's Thesaurus.
2. Open the file containing the subject headings.
3. Read first subject heading into subject_0.
   (a) If (subject_0 is not in CINDI's level_0 subject headings) then Create a level_0 subject heading in CINDI's thesaurus having subject_0's value.
4. Read Next subject heading from the file into subject_1.
   (a) If (subject_1 is not in CINDI's level_1 subject headings having subject_0 as its parent subject) then Create a level_1 subject heading having subject_1's value and subject_0 as its parent subject in CINDI's thesaurus.

5. Read Next subject heading from the file into subject_2

   (a) If (subject_2 is not in level_2 subject headings having both parent subjects subject_0 and subject_1) then create a level_2 subject heading having subject_0 and subject_1 as upper subjects in CINDI's thesaurus.

6. Repeat step five until a flag indicating the end of the corresponding subject_0 and subject_1 hierarchy is reached. If the flag is reached, then we go back to step three. Repeat this procedure until the end of the file is reached.

build-keyword-db.cc This program will construct the controlled term subject association by reading the files that contain a list of root controlled terms and each is followed by the subject headings where it was found. The algorithm followed is:

1. Open CINDI's Thesaurus.

2. Open the file containing the list of controlled terms. Each controlled term is followed by a list of its associated subject headings.

3. Read the control term from the file into key_term.

4. If (The control term is not in CINDI's Thesaurus) then create a new control term having the value key_term in CINDI's Thesaurus.

5. Read next input line into a subject heading.

   (a) If (the subject heading is in CINDI's level_0 subject headings) then add this subject heading into the level_0 subject headings associated with the key_term.

   (b) If (the subject heading is in CINDI's level_1 subject headings) then add this subject heading into the level_1 subject headings associated with the key_term.

   (c) If (the subject heading is in CINDI's level_2 subject headings) then add this subject heading into the level_2 subject headings associated with the key_term.

6. Repeat step five until a flag indicating the end of processing key_term's subject association is reached. If the flag is reached, then go back to step three. Repeat this procedure until the end of the file is reached.

I	you	she	he	it	we	mine
they	me	her	him	us	them	yours
hers	his	its	ours	theirs	their	my
this	that	the	these	those	who	whom
which	what	whoever	whomever	whichever	whatever	all
any	anybody	anyone	anything	each	everybody	everyone
everything	few	many	nobody	none	one	several
some	somebody	someone	myself	yourself	herself	himself
itself	ourselves	a	more	less	also	consequently
finally	furthermore	hence	however	incidentally	indeed	instead
likewise	meanwhile	nevertheless	next	nonetheless	otherwise	still
then	therefore	thus	forever	moreover	only	are
is	afterwards	again	almost	alone	already	always
about	above	across	after	against	along	among
around	at	before	behind	below	beneath	beside
between	beyond	but	by	despite	down	during
except	for	from	in	inside	into	like
near	of	off	on	onto	out	outside
over	past	since	through	throughout	till	to
toward	under	underneath	until	up	upon	with
within	without	amongst	anyhow	anything	anywhere	be
became	become	becomes	becoming	been	beforhand	being
besides	can	and	but	or	nor	for
so	yet	after	although	as	because	before
how	if	once	since	than	that	till
though	until	when	where	whether	while	both
either	neither	whether	an	another		

Table 2: Noise (Stop) words extracted by ASHG

Chapter 5

ASHG

5.1 Introduction

In this chapter, we present the Automatic Semantic Header Generator (ASHG) of the CINDI system. This is an important step in building the Semantic Header. To save time for the document's provider, ASHG provides an initial set of subject classification and a number of components of the Semantic Header for the document. The design goal of ASHG is to automatically build a reliable Semantic Header, which includes classifying a document under a list of subject headings. ASHG's scheme is measuring both the occurrence frequency and positional weight of keywords found in the document. Based on the selected document's keywords, ASHG assigns a list of subject headings by matching those keywords with the controlled terms found in the controlled term subject association.

The ASHG extracts some of the meta-information from a document and stores it in a Semantic Header. For example, when a new document is presented to the system, fields such as document's title, abstract, keywords, dates, author, author's information, size and type are extracted. Using frequency occurrence and positional schemes, ASHG measures the significance of the words found in the previously mentioned list. Word stemming is used in order generate a base form for each word. The system tries to match the base forms of the words with the controlled terms found in

the controlled term subject association. If a match was found the subject headings associated with the controlled terms are extracted and ranked accordingly. The major steps followed by ASHG are briefly described below:

1. *Document Type Recognition*: In order to apply the correct ASHG to a document, the type of the document has to be recognised. The system currently understands HyperText Markup Language (HTML), Latex and plain text documents.

2. *File Type Validation*: The user validates the file type extracted by ASHG.

3. *Applying ASHG's Extractor*: The summariser corresponding to the type of document is applied to the input document.

4. *ASHG's Document Classification*: The document is assigned subject headings. It involves:

   (a) *Word stemming*: The system applies the stemming process[1], to map the words found in the extracted fields onto a base root word.

   (b) *A Look up into the Controlled Term Subject dictionary.*

5. *Semantic Header Validation*: The generated Semantic Header is presented to the user to validate.

5.2 Document Type Recognition

When a document is submitted to the system, the system tries to recognise the type of the document through the name conventions. If it fails the system will then examine its contents. If a failure arises following the examination of the content, the system informs the user that the document type is unrecognised, and the user is asked to enter the Semantic Header. Here are the two steps used in the document type recognition process and which are similar to the Harvest's Essence:

1. Naming conventions and heuristics.

[1] The stemming process will be discussed in more details in section 5.3.5

2. Examining file contents in determining the file types.

The document file upon submitting is passed to a function called *byname*. This function checks the document's name extension. If the extension of the file indicates that it is an HTML, Latex or text, then the function *user_verify* is called. If the naming conventions fails in recognising the document type, the function *bycontent* is called.

```
if (document.extension == .html or .HTML or .htm) then {
   The file is an html file.
   Call function user_verify.
}
else if (document.extension == .tex or .TEX) then {
   The file is a latex file.
   Call function user_verify.
}
else if (document.extension == .doc, or .txt
         or .info or .ascii) then {
   The file is a text file
   Call function user_verify.
}
else {
   Examine the document contents by calling the function bycontent.
}
```

In *user_verify*, the user either confirms or rejects the result. If the user rejects the result, he should choose a type from a list that is displayed. If the user confirms the document's type as recognised, ASHG applies the extractor corresponding to the type confirmed by the user. Otherwise, he should choose a type and then apply ASHG.

In function *bycontent*, the semantics of the HTML and Latex content is exploited when attempting to recognise the file type.

```
If (the file contents match the html file semantics, such as the
    existence of the <HTML> tag) then {
```

```
   the file is of html type
   call function user_verify }
else if (the file contents match the latex semantics, such as the
    existence of the \begin{...} tag) then {
   the file is of latex type
   call function user_verify }
else {
   Unrecognised file type, The user should select a type.
}
```

If the file type is unrecognised, meaning that it is not an HTML, Latex or text, ASHG extracts the size of the file and the date of creation.

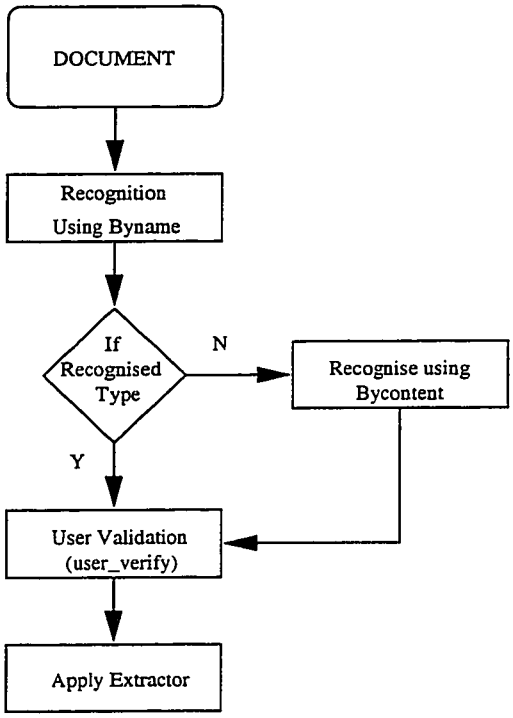

Figure 7: Document Type Recognition

5.3 Applying ASHG's Extractors

Based on the document's type uncovered in the document type recognition step, ASHG applies an extraction procedure. ASHG uses its understanding of HTML, Latex and text syntax documents to extract the document's meta-information. ASHG's *HTML_extractor*, *LATEX_extractor*, *TEXT_extractor* and *UNKNOWN_extractor* are applied to *HTML* type documents, to *Latex* type documents, to *Text* type documents and to unrecognised type document respectively.

Using the document's syntax, ASHG extracts summary information, such as the title, keywords, dates of creation, author, author's information, abstract and size. In both *HTML* and *Latex* documents, the author might explicitly tag some of the fields to be extracted. In case these fields were not explicitly tagged, ASHG attempts to extract them using some heuristics. For example, extracting the keywords in an *HTML* document, The *HTML_extractor* extracts words that are found in the *meta* tag field, if they were included by the author. However, if the explicit keywords were not found in the document, then words found in the title, abstract and other tagged words would be used to extract an implicit list of keywords.

5.3.1 HTML_extractor

HTML is designed to specify the logical organisation of a document, with important hypertext extensions [21]. An HTML document is designed in a way to mark selections of text as titles or paragraphs, and then leaves the interpretation of these marked elements up to the browser. The *HTML_extractor* exploits this mark-up in order to extract the meta-information. HTML instructions divide the text of a document into blocks called elements. These can be divided into two broad categories:

1. the *HEAD* part, which defines information about the document, such as title, and
2. the *BODY* part, which defines how the body of the document is to be displayed by the browser.

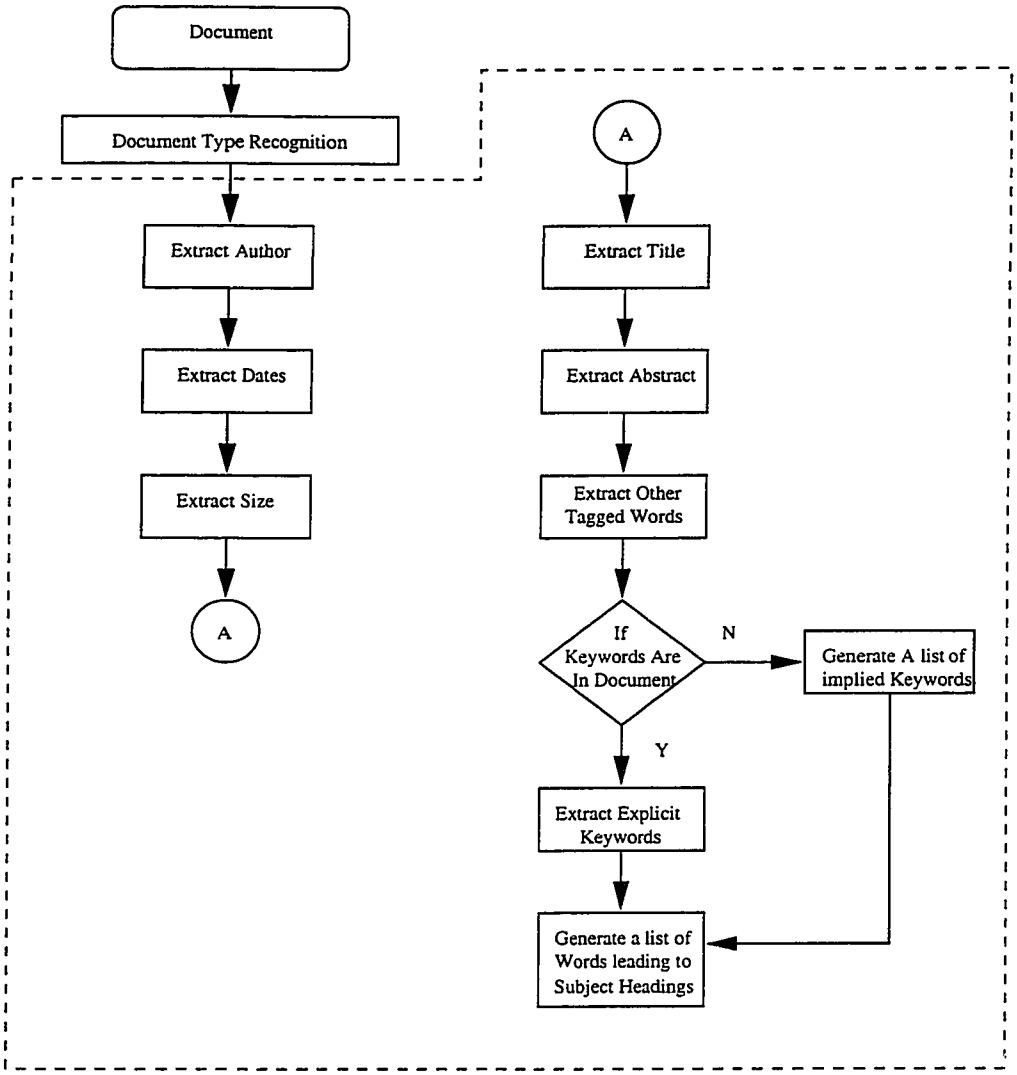

Figure 8: ASHG's extraction steps

ASHG exploits both the *HEAD*'s *TITLE* mark-up element, which is mandatory, and the *HEAD*'s *META* mark-up element, which is a general element for document meta-information. The *META* mark-up element contains information such as date of creation, and date of expiry. It can also contain *Arbitrary User-Specified Information*, which includes information such as keywords, name of the author, and a summary of the document. In case these mark-up elements are not found in the HTML document, ASHG extracts the meta-information by applying some heuristics that will exploit the *BODY* mark-up elements such as the *Hn* headings, *P* paragraphs, *ADDRESS* Address, *Blockquote*, *Lists* and text emphasis.

HTML_extractor extracts the title, explicitly stated keywords, language (English), author(s), dates (Created, Expiry), size of the file, and the abstract from an *HTML* document. Generating an implicit list of keywords will be discussed in sub-section 5.3.5, and the subject headings classification scheme is described in section 5.4. Both procedures are standard for all extractors.

Extracting the author from an HTML document

The *HTML_extractor* extracts the *author* from the *META* mark-up element. For instance, if the HTML document contains <META name = "author" content = "Sami Haddad" >, the *HTML_extractor* extracts *Sami Haddad* as the author of the document.

Extracting dates from an HTML document

Document's creation and expiry dates could be found in the *META* mark-up element, for example <META name = "Created" content = "18/03/98">. The *HTML_extractor* extracts both the date of creation and date of expiry. If it fails to locate them in the *META* mark-up element, it uses the *stat* and *GM-time* commands to extract the date of creation. *stat unix* command contains information about the file such as *File size* in bytes, *Time of last access*, *Time of last data modification* and *Time of last file status change*. *GM-time unix* command converts the time to Coordinated Universal Time (UTC), which is what the UNIX system uses internally.

Extracting the size of the HTML document

Using the *stat* unix command, the size of the file can be extracted.

Extracting the title from an HTML document

The *title*, which is found in between <title> and </title> tags, is extracted. For example, if the HTML document contains <title> *Cindi System analysis* </title>, *HTML_extractor* extracts *Cindi System analysis* as the document's title. If the title tags were not found in the *HTML* document, then the *HTML_extractor* will extract the first heading found in between <h1> and </h1> tags. If it fails, then the first sentence is extracted after generating an HTML tag-free document.

Extracting the abstract from an HTML document

The *HTML_extractor* attempts first to extract the content from the *META abstract* mark-up element. If it fails to find the abstract in the *META* mark-up element, it extracts the paragraph headed by the tagged word abstract. If it fails to locate an abstract heading, it applies an automatic abstracting method. This method, which is similar to Luhn's automatic abstracting method described in chapter 3, attempts to extract a section or paragraph that is headed by *introduction.* Based on the number of significant root words in the sentence, a numerical measure is developed for a sentence. The automatic abstracting includes the highest measured sentences in the abstract. If it fails, the *HTML_extractor* extracts the first paragraph after removing the HTML tags and applies the automatic abstracting method, described above, on this paragraph.

Extracting other tagged words from an HTML document

The *HTML_extractor* extracts a list of tagged words. For example, if the *HTML* document contains the meta tags Database , the *HTML_extractor* includes *Database* in the list of other tagged words. This list of words is used in generating an implied list of keywords and in generating a list of significant words used in the document classification scheme. Both processes will be described in subsection: 5.3.5.

Extracting explicitly stated keywords from an HTML document

The *HTML_extractor* attempts first to extract the *keywords* from the *META* mark-up element. If it fails, it extracts the list of keywords following the tagged word, keywords. For example, if the HTML document contains the meta tags Keywords :*Bibliographic record, search engineering , analysing search, Content description, Database Systems, Expert System, Indexing applications, Searching, URC* <p>, the *HTML_extractor* extracts these as the document's keywords.

5.3.2 Latex_extractor

LaTeX is a TeX macro package, originally written by Leslie Lamport [31], that provides an easy way to use the TeX document processing system. LaTeX allows mark-up to describe the structure of a document, so that the user need not think about presentation. By using document classes and add-on packages, the same document can be produced in a variety of different layouts. LATEX is a macro package which enables authors to typeset and print their work at the highest typographical quality, using a predefined, professional layout.

Latex commands describe the structure of the document. There is a list of things that should be realized about these commands:

1. All Latex commands consist of a backslash followed by one or more characters.
2. Latex commands should be typed using the correct mixture of upper and lower case letters.
3. Some commands are placed within the text. These are used to switch on and off things, like different type styles.
4. There are other commands that look like \command{text}
5. When a command's name is made up entirely of letters, its end is marked by something that isn't a letter. The mark, for instance, could be a space.

The *LATEX_extractor* exploits the use of mark-up elements of specifically the Latex article style to extract the meta-information. It extracts the title, explicitly stated keywords, language (English), author(s), dates (Created, Expiry), size of the file, and the abstract from a *Latex* document. Generating both implicit keywords and a list of subject headings for a document will be described in a later section, since they are a standard procedure for all extractors.

Extracting the author from a Latex document

The \author{names} command declares the author(s), where the name(s) is a list of authors separated by *and* commands. The \\ is used to seperate lines within a single author's entry. For example, to give the author's institution or address. If the following was in the latex document:

```
\author{Bipin C. DESAI \\
Department of Computer Science,\\
 Concordia
University, \\
Montreal, H3G 1M8, CANADA\\}
```

LATEX_extractor extracts *Bipin C. DESAI* as the author's name and *Department of Computer Science, Concordia University, Montreal, H3G 1M8, CANADA* as the author's address.

Extracting dates from a Latex document

The \date{text} declares *text* to be the document's date. For example, if "\date{*18/04/98*}" was found in the Latex document, *LATEX_extractor* extracts *18/04/98* as the document's date. If no \date command is found in the Latex document, *LATEX_extractor* uses the *stat* and *GM-time* commands to extract the date of creation.

Extracting the size of the Latex document

Using the *stat* unix command, the size of the file can be extracted.

Extracting the title from a Latex document

The \title{text} command declares text to be the title. The *LATEX_extractor* exploits the title mark-up element to extract the title. For instance, if the latex document contains \title{CINDI system analysis}, *LATEX_extractor* extracts *CINDI system analysis* as the title. If it fails, *LATEX_extractor* exploits the following mark-up element:

```
\begin{titlepage}
   text
\end{titlepage}
```

It extracts the first sentence found in the *text*. If it fails, it extracts the text marked up by huge, or large. It can exploit the presence of \begin{huge} *text* \end{huge} or \huge{*text*}. If it fails, it exploits the presence of \begin{Large} *text* \end{Large} or \Large{*text*}. If none of the above mark-up elements were found in the document, *LATEX_extractor* filters out all latex mark-up elements and extracts the first sentence as the document's title.

Extracting the abstract from Latex document

A latex document might contain \begin{*abstract*} *text* \end{*abstract*}. If it does, the *LATEX_extractor* extracts the *text* as the document's abstract. Otherwise, it extracts the sections which are headed by the word *abstract*. For example, if \section{*Abstract*} is found in the document, the paragraph that follows is extracted.

However, if it fails, it extracts the paragraph that follows \huge{*Abstract*}, \large{*Abstract*}, \bf{*Abstract*}, or \it{*Abstract*}. If none of these are found, the automatic abstracting method is applied. This method, which is similar to Luhn's automatic abstracting method, is described in chapter 3 and in the *HTML_extractor*. If the automatic abstracting method fails, the first marked up paragraph is extracted, otherwise, all the latex mark-ups are removed and the first paragraph is extracted as the document's abstract.

Extracting other tagged words from a latex document

The *LATEX_extractor* extracts a list of other marked up words. It uses the sectioning commands and the three typefaces latex commands: \em (Emphatic), \bf (Boldface) and \it (Italic) to extract the marked up words. The extracted words will be used in the generation of an implicit list of keywords and the generation of a list of significant words used in the document's classification scheme. This process of generating an implicit list of keywords and a list of significant words is described in subsection 5.3.5. Here are the Latex sectioning commands:

- \part
- \chapter
- \section
- \subsection
- \subsubsection
- \paragraph
- \subparagraph

Extracting explicitly stated keywords from a latex document

LATEX_extractor exploits three typefaces latex commands: \em (Emphatic), \bf (Boldface) and \it (Italic). These commands are used inside a pair of braces to limit the amount of text that they affect. For instance, if the following was in the latex document:

{*bf Keywords:* } *Information retrieval, Modelling, meta-data, cataloguing. searching, discovery, information resources, WWW, Internet, resource discovery* \\

, the *LATEX_extractor* extracts the words as the document's keywords.

5.3.3 Text_extractor

Perhaps one of the most challenging tasks in information extraction is to extract and manipulate information found in plain text documents. Since these documents do not contain tags or mark-up elements, the *TEXT_extractor* relies heavily on heuristics in extracting the title, explicitly stated keywords, language (English), author(s), dates (Created, Expiry), size of the file, and the abstract from a *Latex* document. Generating both implicit keywords and a list of subject headings for a document will be described in a later section.

Extracting the author from a plain text document

The *TEXT_extractor* looks for a pattern such as *written by, edited by* or *revised by.* If it finds one of them, it extracts the text following it and stores it as the author's Semantic Header field.

Extracting dates from a plain text document

The *TEXT_extractor* uses the *stat* and *GM-time* commands on the document file to extract the date of creation.

Extracting the size of the plain text document

Using the *stat* unix command, the size of the file can be extracted.

Extracting the title from a plain text document

When presented with a plain text document, the *TEXT_extractor* extracts the first sentence from the document. This sentence is used as the document's title. If it fails, it generates a list of sentences by extracting all sentences found in the first, second and last paragraph and by extracting the first sentence of all other paragraphs. Each sentence is divided into its constituent words. After dropping all English Noise or Stop words, the remaining words are stemmed.

Each sentence is given a weight according to the frequencies occurrences' sum of the stemmed words found in the sentence. The *TEXT_extractor* selects the highest weighted sentence as the document's title.

Extracting the abstract from plain text document

The *TEXT_extractor* looks for the pattern, *abstract*, and extracts the first paragraph following it. If it fails, the automatic abstracting method is applied on the document's introduction . If it fails to construct an abstract, *TEXT_extractor* applies the automatic abstracting method on the sentences found in the first, second and last paragraph and on the first sentence of all other paragraphs. The sentences are divided into their constituent words. Dropping all English Noise words, the remaining words are stemmed. The extracted sentences are weighted according to the frequency occurrence of the stemmed words. The *TEXT_extractor* will construct the document's abstract by extracting the highest weighted sentences.

Extracting other words from a plain text document

The *TEXT_extractor* extracts the words found in the first two paragraphs, the last paragraph and in the first sentence of each other paragraph. After removing the English Noise words, a list of stemmed words is derived. The derived words will be used in the generation of an implicit list of keywords and the generation of a list of significant words used in the document's classification scheme.

Extracting explicitly stated keywords from a plain text document

The *TEXT_extractor* extracts the text following the word *keyword* as the document's keywords, until the *TEXT_extractor* reaches an *introduction* heading or a new paragraph.

5.3.4 Unknown_extractor

ASHG supports *HTML, Latex* and *Text* documents; however, if the document is not any of these types, ASHG applies the *UNKNOWN_extractor*. It extracts the *size* of the document and the *creation*

date. It is up to the document's author or provider to enter the Semantic Header's information.

5.3.5 Generating an implicit list of keywords and words used in Document classification

ASHG generates an implicit list of keywords in case explicit keywords were not found in the document. It derives a list of most significant words, which is used in the document classification scheme.

In case keywords were not found in the document, the system derives a list of words from the words found in the title, abstract, and other tagged fields. This list of derived words will also be used in classifying the document. However, if the keywords were explicitly stated in the document, then ASHG will generate a list of words from the words found in the title, abstract, keywords and other tagged fields. This list is used to assign a list of subject headings for the document.

Generating both lists of words relies on the stemming process that will map the words into their root words, the stemmed word frequency of occurrence and the word location in the document. It uses the following algorithm in generating the list of implicit keywords, in case the keywords were not found in the document, and the words used in the classification scheme:-

1. Extract the title, abstract and other tagged fields. If the document wasn't tagged such as in a plain text document, words found in the first two and last paragraphs and in the first sentence of each paragraph are selected. Keywords are extracted if they were found in the document.
2. English Noise words constitute usually around 30 to 50 per-cent of a document. The Information Retrieval community calls them the *Stop List*. These words are dropped from the extracted fields.
3. The remaining words are sent to the stemming process. This process will remove the words' suffixes and prefixes. For example, the words: *cycled, cycler, cycling* and *cycles* are stemmed to the root term, cycle. The aim of the stemming process is to generate base word class, which include all the forms that could be generated from it.

4. Because the terms are not equally useful for content representation, it is important to introduce a term weighting system that assigns high weights for important terms and low weight for the less important terms [55]. Therefore, the weights constitute the importance of a word. The system assigns weights to both lists of root words. The weight assignment uses the following scheme:

   (a) If the word appears in the explicitly stated keywords, it is assigned a weight of five [2]. Since authors explicitly state the keywords to convey some important terms, which their document covers, it is assigned the highest weight. For example, if the word *device* is found in the list of explicitly stated keywords, the word *device* is assigned a weight of five.

   (b) Usually, words found in the abstract are the second most important words, because this is where the author tries to convey his/her idea. Therefore, words found in the abstract are the second most significant and they convey the idea of the article more than any words found in other locations [51]. If the word appears in the abstract, it is assigned a weight of four.

   (c) If the word appears in the title, it is assigned a weight of three. For example, if the word *compute* is found in the title, it is assigned a weight of three.

   (d) If the word appears in the other tagged words, it is assigned a weight of two.

5. Each numeric weight is a class by itself defining the words' location. The system has the following classes:

   (a) A class weight of two defines the OTHER WORDS class. This class contains the terms found in only the OTHER WORDS field.

   (b) A class weight of three defines the TITLE class. The class three contains all the terms found only in the title field.

   (c) The class weight four contains all the terms found only in the abstract field. Therefore a class weight of four defines the ABSTRACT class.

   (d) A class weight of five includes all the terms found in either the keywords' field or in the title and other words fields.

   (e) A class weight of six includes all the terms found in both the abstract field and the other words field.

   (f) A class weight of seven includes all the terms found in either the keyword and other words fields or the abstract and title fields.

[2] If the keywords are stated, then they will be used in addition to the other weight classes in determining the subject classification for the document

(g) A class weight of eight contains all the terms found in keyword and title fields. For example, if the word *compute* appears in both the title and explicitly stated keywords, it is assigned a weight of eight. The word *compute* will be an element of the class weight of eight.

(h) A class weight of nine contains all the terms found in either the abstract, title and other words fields, or abstract and keywords fields.

(i) A class weight of ten contains all the terms found in the other words, title and keywords fields.

(j) A class weight of eleven contains all the terms found in the other words, abstract and keywords fields.

(k) A class weight of twelve contains all the terms found in the title, abstract and keywords fields.

(l) A class weight of fourteen contains all the terms found in the other words, title, abstract and keywords fields.

A term appearing in other words field is less important than the one appearing in the abstract field. Furthermore, a term appearing in both title and other words fields is less significant than the one appearing in the keywords, abstract and title field. In a high class weight, we are interested in extracting more terms than in lower class weights. Thus, we tend to extract more terms from the high weighted classes. To limit the number of extracted terms, we use the term's frequency of occurrence.

Significant terms have the highest frequency of occurrence in the low weighted classes. As the class weight increases, more of its terms are regarded as significant. To include more significant terms, the domain of the terms' frequencies is expanded. The more is the class weight, the wider is the domain frequency of the significant terms.

For each class, we set the maximum class frequency to be the maximum frequency of occurrence of a term found in that class. For instance, if, in class four, we had three terms having two, four and six as frequencies, the system would select six as the maximum class four frequency.

The words' frequencies are compared with their corresponding maximum class frequency. For low weighted classes such as two and three, significant terms have the maximum class

frequencies. Thus, limiting the number of significant terms. However, all terms found in class eight and more are significant regardless of their frequency of occurrence.

Term Weight	Term Frequency
2	Maximum Class 2 frequency
3	Maximum Class 3 frequency
4	Greater or equal to Maximum Class 4 frequency minus 1
5	Greater or equal to Maximum Class 5 frequency minus 1
6	Greater or equal to Maximum Class 6 frequency minus 2
7	Greater or equal to Maximum Class 7 frequency minus 3
8 or more	All

Table 3: Weight and Frequency numbers used in extracting terms

6. Two lists of words will be generated. The first one containing only the root words or controlled terms found in CINDI's thesaurus. This list of controlled terms is used in the document's subject classification scheme. The second list contains the most significant root words not found in CINDI's thesaurus.

7. If no keywords were found in the document, ASHG extracts words having a term weight more than four and their corresponding frequencies of occurrence is the same as the ones tabulated. These words are the document's keywords.

8. In generating a list of controlled terms used to classify the document, terms having weight of two or more are extracted. The extracted words should have the frequencies of occurrence as the ones tabulated.

ASHG's Stemming Process

Stemming consists of processing a word so that only its stem or root form is left. In Okapi [74], indexed keywords are held in stemmed form, and so are query words entered by the user. This provides a better likelihood of matching relevant documents. In the Okapi [74] system, a stemming algorithm developed by Porter [44] at Cambridge is used. This algorithm uses weak stemming to remove common plural endings and other grammatical suffixes like *-ing* and *-ed* and implements strong stemming to remove derivational suffixes like *-ent*, *-ence*, and *-ision*.

Many searchers use right hand truncation to find different variations of a search term that is of interest. For example, rather than search for ultrasonic, ultrasonically or ultrasonics and cleaner, cleaning, cleaners, cleaned or cleanable, a searcher will right hand truncate the terms and retrieve all terms sharing the specified root ultrasonic and clean. The problem with right hand truncation is that it indiscriminately adds words to the query [75].

For example, if a searcher were to search for the truncated form of the word *cover*, the searcher would not only retrieve instances of the terms *covers, covering* and *covered* but also the terms *covert, coverall, coversheet* and *coverage*. QPAT-US [75] helps you avoid extraneous right hand truncation terms by automatically performing a process called *stemming*. First, QPAT-US evaluates your terms for common suffixes that indicate plurality, verb tense, etc. If QPAT-US discovers these suffixes, it will strip them to find the root form of the term. For instance, if QPAT-US finds the term *covering* it will strip the suffix to obtain the root word of *cover*. Next, QPAT-US takes the root form of your search terms and, using sophisticated linguistic rules, creates a set of word variants. If your original term is *covering*, QPAT-US will also search for *cover, covers* and *covered*.

Two types of stemming are available in various versions of WAIS: Porter and Plural. Plural stemming attempts to identify and index the singular form of a term. Porter stemming attempts to identify and index the word *stem*. If a word and its stem are different, only the word stem is indexed.

ASHG's stemming process implements the removal of both suffixes and prefixes of a given word in order to get the root of the word. For example, applying the stemming process on the words *simulation* and *analogies*, the words *simulate* and *analogy* are generated as their root words respectively. ASHG stores the root forms of the words.

Suppose the word impressionists is in a document for which meta-information is to be extracted. Without stemming, this would match only the keyword impressionists and not the singular form. Now suppose that the word impressionist was in CINDI's list of controlled terms, then that document

will miss that term and will not have it as a keyword. Following stemming, documents having the word impressionistic and impressionism will match the root term that is found in CINDI's list of controlled terms.

We have mainly used the *spell* unix command in our system in extracting the root of a word. The *spell* command collects words from an input file and looks them up in a dictionary list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. Two options were used in our system:

1. *-v* All words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.
2. *-x* Every plausible stem is displayed, one per line, with = preceding each word.

The steps of the ASHG stemming process are:

1. Using the *sort* unix command, sort the input words.
2. Apply the *uniq* unix command to filter out duplicate words.
3. Apply the *spell* command with the *-x* option. Thus, all the plausible stems are stored in an output file.
4. Apply the *spell* command with the *-v* option. Thus, all words not found in the spelling list are stored.
5. Create a file which contains the words found in step 3 but not in step 4.
6. Apply the *spell* command with the *-v* option to each word found in the file that resulted from the previous step. If the resulting output is empty, this means that this root word is found.

We applied the ASHG's stemming process and Porter's algorithm to a set of words and compared the resulting root words. The results are shown in the following table:

Words	Porter's algorithm	ASHG algorithm
Adventure	adventur	adventure
games	game	game
Computer	comput	compute
aided	aid	aide
engineering	engine	engineer
Transcription	transcript	transcription
Algorithms	algorithm	algorithm
Animation	animat	animate
construction	construct	construct
industries	industry	industry
Industrial	industry	industry
analogies	analogy	analogy
Mathematical	mathemat	mathematic
models	model	model
Simulation	simul	simulate
Civilization	civil	civil
COMPLEXITY	complex	complex
assisted	assist	assist
Authoring	author	author
graphics	graphic	graphic
programmers	programm	programmer
programming	program	programming
Computerized	computer	compute
Performance	perform	performance
Coding	code	code
Utilities	utility	utility
Optimization	optimiz	optimize
Combinatorics	combinator	combinatoric
Representation	represent	presentation

Table 4: The word stem results after using Porter and ASHG's Algorithm

5.4 ASHG's Document Subject Headings Classification scheme

An important step in constructing the semantic header is to automatically assign subject headings to the documents. The title, explicitly stated keywords, and abstract are not enough by themselves to convey the ideas or subjects of the document. Since the author tries to convey or to summarise his ideas in the previously mentioned fields, there is a need to use all English none noise words found in those fields. To assign the subject headings, ASHG uses the resulting list of significant words generated from the previous section and CINDI's controlled term subject association. The subject heading classification scheme relies on passing weights from the significant terms to their associated subjects, and selecting the highest weighted subject headings.

5.4.1 The Algorithm followed

Having the keywords, title words, abstract words and other tagged words, will help us select the most appropriate subjects for a given document. The following algorithm is used:-

1. Three lists of subject headings are to be constructed. The list of Level_0 subject headings, the list of Level_1 subject headings and the list of Level_2 subject headings.
2. For each term found in both CINDI's controlled terms and the generated list of words, the system traces the controlled term's attached list of subjects (list of *level0, level1 and level2*) headings, and adds the subject headings to their corresponding list of possible subject headings.
3. Weights are also assigned to the subject hierarchies. The weight for a subject is given according to where the term matching its controlled term was found. A subject heading having a term or set of terms occuring in both title and abstract, for instance, gets a weight of seven. The matched terms' weights are passed to their subject headings.
4. The system extracts *Level_2, Level_1* and *Level_0* subject headings having the highest weights from the three lists of possible subject headings.
5. After building the three lists for the three level subject headings, the system :
   (a) selects the subjects using the bottom-up scheme.
   (b) Having selected the highest weighted *level_2* subject headings, the system derives their *level_1* parent subject headings.

(c) An intersection is made between the derived *level_1* subject headings and the list of the highest weighted *level_1* subject headings. The common *level_1* subjects are the document's *level_1* subject headings.

(d) The system uses the same procedure in selecting *level_0* subject headings.

5.5 Semantic Header Validation

Once the process of extracting the meta-information is terminated, the semantic header is displayed for the source provider to modify, add or remove some of the attributes. Once the provider finishes, the semantic header can be stored in the CINDI database.

Chapter 6

Analysis of ASHG's Results

In this chapter, we illustrate how the ASHG system extracts the meta-information from the HTML, Latex and text documents, and we demonstrate ASHG's automatic subject headings classification. For each of these document types, we apply ASHG and show the results. We compare the subject classification generated by ASHG with that of INSPEC for the same set of documents. We also compare the results with what the papers' authors would regard as good subject classifications and poor ones.

6.1 Reduction of Controlled Terms

Salton et al [55] introduces the term weighting system that assigns high weights to terms deemed important and lower weights to the less important terms. The term weighting system favours terms with high frequency in particular documents but with a low frequency overall in the collection.

ASHG's controlled terms favours the terms that have low frequency in the ASHG's subject headings over the terms having high frequency. Controlled terms having high frequency are dropped from the ASHG's list of controlled terms. Terms having lower frequency distinguish the subject headings associated for the document.

The controlled term *system* occurs two hundred and eleven times in the ASHG's subject headings, which is the highest frequency control term. Therefore, it is dropped from the ASHG's list of controlled terms. The following table shows the words that are dropped and their corresponding frequencies.

Words	Frequency
system	211
power	115
design	106
electric	100
circuit	96
application	93
language	87
device	84
measure	83
general	72
manage	71
information	70
analysis	69
miscellaneous	58
other	47

Table 5: Words Dropped from the list of controlled terms

Other control terms such as section, two, three, function, and method were dropped due to their ambiguity.

6.2 Experiments

The experiments described here are designed to test the accuracy of the generated index and the subject headings classification results. After applying the ASHG on a set of documents, the generated index fields such as title, keywords, abstract and author are compared with those that are found in the document. The ASHG's automatic subject headings classification results are compared with the INSPEC's classification and with what the papers' authors would regard as good subject classifications and poor ones.

The experiments were conducted on thirty three documents. The titles of these documents can be viewed in appendix A. These documents dealt with computer science and electrical engineering subjects. ASHG was able to extract all the explicitly stated fields such as title, abstract, keywords, and author's information with a hundred percent accuracy. If the abstract was not explicitly stated, ASHG was able to automatically generate an abstract that would describe the paper. However, ASHG's implicit keyword extraction generated a list of words which included some words that are insignificant. These insignificant words in turn lead to the diversion in subject classification.

We have consulted the papers' authors on the ASHG's subject classification results. Their response was divided into three categories: *good, OK/Not sure* and *poor* subject hierarchy selection. Good subject hierarchy selection implied that the authors would have chosen them as subject hierarchies for the documents. *OK/Not sure* subject hierarchy selection implied that the authors doubt the results and they would not choose them. Finally, the *poor* subject hierarchy selection implied that the selected subject hierarchies described another different subject.

We compared the ASHG's subject classification results against the INSPEC's classification done by expert cataloguers and thesaurus. Some of the ASHG's subject classification had different words than INSPEC's even though they described the same subject. That was due to the fact that our computer science subject classification was built from ACM and not from INSPEC.

After conducting the tests over the three document types, ASHG's average percentage accuracy was 21.92%. Since our system was only based on the frequency and location of words in a document to determine the document's keywords and subject classification, it has missed the importance of the word senses and the relationship between words in a sentence. Our simplistic system did not capture the concepts behind the documents, or the ideas that the author is trying to convey.

Our results support the idea that word frequency and location are not enough in information retrieval. However, since the ASHG's result will be used as a starting point by the author, he/she

HTML Document	Number of Subject Headings generated by ASHG	Author's Opinion			Accuracy A: Author I: INSPEC	OK/Good's Accuracy
		Good	OK/Not Sure	Poor		
D1	6	4	2	0	66.66% (A) 16.6% (I)	100%
D2	7	4	1	2	57.14% (A)	71.42%
D3	8	6	0	2	75% (A)	75%
D4	9	3	4	2	33.33% (A)	77.77%
D5	7	0	3	4	0 (A)	42.85%
D6	6	0	2	4	0 (A)	33.33%
D7	4	1	3	0	25% (A)	100%
D9	6	0	3	3	0 (A) 16.66% (I)	50%
D10	5	0	4	1	0 (A)	80%
D11	3	0	1	2	0 (A)	33.33%
D12	5	1	4	0	20% (A) 20% (I)	100%
D13	5	1	3	1	20% (A)	80%
D14	5	0	3	2	0 (A)	60%
D15	6	1	4	1	16.66% (A) 16.66% (I)	83.33%
D17	3				0 (I)	
D18	3	1	1	1	33.33% (A) 33.33% (I)	66.66%
D19	7	4	1	2	57.14% (A) 42.8% (I)	71.42%
D20	4	1	0	3	25% (A)	25%
D21	5	0	2	3	0 (A) 20% (I)	40%
D22	4				25% (I)	
D23	6				16.66% (I)	
D24	4				25% (I)	
D25	4				25% (I)	
D26	3				0% (I)	
D27	3				66.66% (I)	
D28	26				7.69% (I)	
D29	5				0 (I)	
D30	3				0% (I)	
D31	5				20% (I)	
D32	5				40% (I)	
Averages					22.2%	66.11%

Table 6: Summary of ASHG's HTML test results against the authors and INSPEC's results

Latex Document	Number of Subject Headings generated by ASHG	Author's Opinion			Accuracy A: Author I: INSPEC	OK/Good's Accuracy
		Good	OK/Not Sure	Poor		
D1	5	2	3	0	40% (A) 16.6% (I)	100%
D2	5	4	0	1	66.66% (A)	66.66%
D3	4	1	0	3	25% (A)	25%
D4	10 from 11	4	4	2	40% (A)	80%
D5	6	0	2	4	0 (A)	33.33%
D6	4	0	2	2	0 (A)	50%
D7	4	0	2	2	0 (A)	50%
D8	5	1	2	2	20% (A)	60%
D9	4 from 6	0	2	2	0 (A) 25% (I)	50%
D10	4	0	4	0	0 (A)	100%
D11	5	1	1	3	20% (A)	40%
D12	5	0	5	0	0 (A) 20% (I)	100%
D13	6	2	3	1	33.33% (A)	83.33%
D14	4	0	4	0	0 (A)	100%
D15	5	1	4	0	20% (A) 20% (I)	100%
D16	4	1	2	1	25% (A)	75%
D17	3				0 (I)	
D18	3	1	1	1	33.33% (A) 66.66% (I)	66.66%
D19	3	1	0	2	50% (A) 50% (I)	50%
D20	4	1	0	3	25% (A)	25%
D21	3	0	0	3	0 (A) 33.33% (I)	0
D22	7				28.57% (I)	
D23	7				14.28% (I)	
D24	4				0 (I)	
D25	24				25% (I)	
D26	4				0% (I)	
D27	3				66.66% (I)	
D28	26				50% (I)	
D29	5				0 (I)	
D30	4				50% (I)	
D31	25				4% (I)	
D32	4				25% (I)	
Averages					22.91%	62.75%

Table 7: Summary of ASHG's Latex test results against the authors and INSPEC's results

Text Document	Number of Subject Headings generated by ASHG	Author's Opinion			Accuracy A: Author I: INSPEC	OK/Good's Accuracy
		Good	OK/Not Sure	Poor		
D1	5	2	3	0	40% (A) 20% (I)	100%
D2	17	1	3	13	5.88% (A)	23.52%
D3	8	6	0	2	75% (A)	75%
D4	4	1	2	1	25% (A)	75%
D5	7	0	3	4	0 (A)	42.87%
D6	6	0	2	4	0 (A)	33.33%
D7	5	1	3	1	20% (A)	80%
D8	5	2	2	1	40% (A)	80%
D9	4	0	2	2	0 (A) 0 (I)	50%
D10	7	0	2	5	0 (A)	28.57%
D11	9	4	3	2	44.44% (A)	77.77%
D12	5	1	4	0	20% (A) 20% (I)	100%
D13	5	2	2	1	40% (A)	80%
D14	7	0	3	4	0 (A)	42.85%
D15	4	1	3	0	25% (A) 20% (I)	100%
D16	7	3	1	3	42.85% (A)	57.14%
D17	3				0 (I)	
D18	3	1	0	2	33.33% (A) 33.33% (I)	33.33%
D19	5	1	1	3	20% (A) 20% (I)	40%
D20	5	1	0	4	20% (A)	20%
D21	3	0	0	3	0 (A) 0% (I)	0
D22	4				50% (I)	
D23	8				12.5% (I)	
D24	4				25% (I)	
D25	28				7.14% (I)	
D26	4				25% (I)	
D27	44				25% (I)	
D28	4				0 (I)	
D29	18				27.77% (I)	
D30	14				14.28% (I)	
D31	28				14.28% (I)	
D32	5				40% (I)	
Averages					20.66%	56.97%

Table 8: Summary of ASHG's Text test results against the authors and INSPEC's results

has the opportunity to correct the errors and include fields of the Semantic Header not given before registering it. Further work is required in refining the subject classification.

6.2.1 Sample Results

In this section, we will show some of the indexes generated by ASHG.

```
<semhdrB>
<useridB> <useridE>
<passwordB> <passwordE>
<titleB> Resource Discovery: Modelling, Cataloguing and Searching <titleE>
<alttitleB> <alttitleE>
<subjectB>
<generalB> Computer Science <generalE>
<sublevel1B> Software <sublevel1E>
<sublevel2B> Computer programs and softwares <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Information storage and retrieval <sublevel1E>
<sublevel2B> Information search and retrieval <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Information storage and retrieval <sublevel1E>
<sublevel2B> Query formulation in information search and retrieval <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Information storage and retrieval <sublevel1E>
<sublevel2B> Relevance feedback in information search and retrieval <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Information storage and retrieval <sublevel1E>
<sublevel2B> Retrieval models in information search and retrieval <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Information storage and retrieval <sublevel1E>
<sublevel2B> Information search and retrieval process <sublevel2E>
<subjectE>
<languageB> English <languageE>
<char-setB> <char-setE>
<authorB>
<aroleB> Author <aroleE>
<anameB> Bipin C. DESAI, "" ""Rajjan SHINGHAL <anameE>
<aorgB> <aorgE>
<aaddressB>  Department of Computer Science, Concordia University, Montreal,
             H3G 1M8, CANADA <aaddressE>
<aphoneB>  <aphoneE>
<afaxB>  <afaxE>
<aemailB>  <aemailE>
<authorE>
<keywordB>  Information retrieval , Modelling , meta-data , cataloguing
            searching , discovery , information resources , WWW , Internet ,
            resource discovery  <keywordE>
<identifierB>
<domain3B> <domain3E>
<value3B> <value3E>
```

<identifierE>
<datesB>
<createdB> 1998/4/18 <createdE>
<expiryB> <expiryE>
<datesE>
<versionB> <versionE>
<spversionB> <spversionE>
<classificationB>
<domain4B> <domain4E>
<value4B> <value4E>
<classificationE>
<coverageB>
<domain5B> <domain5E>
<value5B> <value5E>
<coverageE>
<system-requirementsB>
<componentB> <componentE>
<exiganceB> <exiganceE>
<system-requirementsE>
<genreB>
<formB> <formE>
<sizeB> 42301 <sizeE>
<genreE>
<source-referenceB>
<relationB> <relationE>
<domain-identifierB> <domain-identifierE>
<source-referenceE>
<costB> <costE>
<abstractB>
Existing search systems exhibit uneven selectivity when used in seeking information resources on the Internet. This problem has prompted a number of researchers to turn their attention to the development and implementation of meta-data models for use in indexing and searching on the WWW and Internet. In this paper, we present our re-sults of a simple query on a number of existing search systems and then discuss a pro-posed meta-data structure. Modelling the expertise of librarians for cataloguing, user entry and search using a rule-based system is also discussed.
<abstractE>
<annotationB>
<annotationE>
<semhdrE>
<EOF>

```
<semhdrB>
<useridB> <useridE>
<passwordB> <passwordE>
<titleB> On the Accuracy of the UTD for the Scattering by a Cylinder <titleE>
<alttitleB> <alttitleE>
<subjectB>
<generalB> Electrical Engineering <generalE>
<sublevel1B> Communications <sublevel1E>
<sublevel2B> Radar equipment, systems and applications <sublevel2E>
<generalB> Electrical Engineering <generalE>
<sublevel1B> Electromagnetic waves, antennas and propagation
             fields <sublevel1E>
<sublevel2B> Electromagnetic wave propagation <sublevel2E>
<subjectE>
<languageB> English <languageE>
<char-setB> <char-setE>
<authorB>
<aroleB> Author <aroleE>
<anameB> Robert Paknys <anameE>
<aorgB> <aorgE>
<aaddressB>  <aaddressE>
<aphoneB>  <aphoneE>
<afaxB>  <afaxE>
<aemailB>  <aemailE>
<authorE>
<keywordB>  scat , case  <keywordE>
<identifierB>
<domain3B> FTP <domain3E>
<value3B> <value3E>
<identifierE>
<datesB>
<createdB> 1998/5/21 <createdE>
<expiryB>  <expiryE>
<datesE>
<versionB> <versionE>
<spversionB> <spversionE>
<classificationB>
<domain4B> <domain4E>
<value4B> <value4E>
<classificationE>
<coverageB>
<domain5B> <domain5E>
<value5B> <value5E>
<coverageE>
<system-requirementsB>
<componentB> <componentE>
<exiganceB> <exiganceE>
<system-requirementsE>
<genreB>
<formB> <formE>
<sizeB> 18870 <sizeE>
<genreE>
```

<source-referenceB>
<relationB> <relationE>
<domain-identifierB> <domain-identifierE>
<source-referenceE>
<costB> <costE>
<abstractB>
The UTD formulation for the scattering by a cylinder is valid for antennas that are removed from the cylinder surface. The usual guideline is that reliable results can be obtained for antennas that are about lambda4 or more away from the surface. By exploring a few cases, we show that lambda4 is unnecessarily large for the lit region and sometimes too small for the shadow region. In addition, we find that with a simple heuristic modification to the UTD, heights as small as lambda20 can be accommodated, with an accuracy that is sufficient for most engineering applications.
<abstractE>
<annotationB>
<annotationE>
<semhdrE>
<EOF>

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we constructed CINDI's three level subject hierarchy for Computer Science and Electrical Engineering. CINDI's computer science subject hierarchy was based on ACM and CINDI's electrical engineering subject hierarchy was based on INSPEC. LCSH was used to augment both subject hierarchies. We also derived control terms from CINDI's subject headings. These control terms were associated with their subjects in CINDI's thesaurus. In addition, we presented a method of generating a Semantic Header, called ASHG. This scheme automatically extracts and generates an index or meta-information.

ASHG exploits the file naming conventions and the data within a document to determine the document's file type. ASHG exploits the semantics of the document's types in extracting the meta-information. It also applies automatic abstracting proposed by Luhn in generating document's abstract. It also assigns weights for terms depending on their location in the document. Both term weight and occurrence frequency were used in assigning terms for a document. These extracted terms were used to classify a document using the association between CINDI's controlled term and their subject headings in the thesaurus.

Finally, we applied ASHG to a collection of test documents and compared the results to the actual assignments made by INSPEC. We also consulted the papers' authors on ASHG's subject classification results. The results showed hundred percent accuracy in extracting the explicitly stated fields such as the title, abstract, author and keywords. They also showed some level of accuracy in generating the abstract.

Since our controlled terms were composed of terms found in CINDI's subject headings, ASHG's results showed a low degree of accuracy in classifying a document. The main reason was that some of the extracted terms were misleading. For example, the term *wire* should not be extracted unless it is followed by another term such as *wire grid.* The classification scheme used by ASHG showed some ineffectiveness, because it was based on term frequency and location information. For example, term-based retrieval cannot handle the following properties:

1. Different words may be used to convey the same meaning.
2. The same words may be used but they can have different meanings.
3. Different people may have different perspectives on the same single concept.
4. The same words may have different meanings in different domains.

Another weakness with ASHG is that it has not considered the issue of synonymity between words or between the subject headings.

In conclusion, we believe that resolving word senses and determining the relationships that those words have to one another will have the greatest impact on refining the ASHG's subject classification scheme. Therefore, the semantic level language processing should be handled by ASHG in the future.

7.2 Contribution of this Thesis

The contributions made by this thesis to the CINDI project are listed below:

- An automatic subject hierarchy database builder was designed and built. The input to this builder is a subject headings of multiple levels. It produces a hierarchy of three levels. In addition, the subject hierarchy for both Computer science and Electrical engineering were constructed and derived from previously existing hierarchies such as ACM and INSPEC.

- A controlled term subject heading association was engineered. The thesis used an existing spell program and built on it a stemming process that was used in relating the subject headings with their corresponding control terms.

- An automatic semantic header generator was designed and implemented. It extracted both implicit and explicit meta-information from the primary resource and it classifies it under a subject hierarchy. It handled HTML, Latex and Text documents.

7.3 Future Work

Some of the system's refinements should include:

- Terms, which are not significant alone, but are significant if they appear adjacent to another term should be extracted as significant terms. ASHG's keyword extraction process should handle more than single controlled terms. Future work should explore the effect of extracting noun phrases and compound controlled terms.

- Word senses and determining the relationships that those words have to each other should be resolved. The semantic level language processing should be handled by ASHG.

- The controlled terms and their synonyms should belong to the same control term and they should be associated with the same subject headings.

- The domain of the stop-word list should be explored, and more significant terms should be associated with the subject headings.

- Build more subject hierarchies such as Civil Engineering, Mechanical Engineering... Extend the type of documents that ASHG can extract meta-information from, such as RTF, SGML...

Appendix A

Papers Used in Testing ASHG

The following is the list of papers used in testing ASHG:

D1 Desai B. C., and Shinghal R., *Resource Discovery: Modelling, Cataloguing and Searching*, Department of Computer Science, Concordia University, Montreal, Canada.

D2 Desai B. C., and Shinghal R., and Radhakrishnan T., *An Expert System to Aid Cataloging and Searching Electronic Documents on the World Wide Web*, Department of Computer Science, Concordia University, Montreal, Canada.

D3 Desai B. C., and Shinghal R., *Modeling Expert Search of Virtual/Digital Libraries*, Department of Computer Science, Concordia University, Montreal, Canada.

D4 Grogono P., *Designing for Change*, Department of Computer Science, Concordia University, Montreal, Canada.

D5 Grogono P., *Designing a class library*, Department of Computer Science, Concordia University, Montreal, Canada.

D6 Grogono P., *Design for a Text Editor*, Department of Computer Science, Concordia University, Montreal, Canada.

D7 Grogono P., *A Code Generator for Dee*, Department of Computer Science, Concordia University, Montreal, Canada.

D8 Grogono P., *The Dee Report*, Department of Computer Science, Concordia University, Montreal, Canada.

D9 Butler G., Grogono P., Shinghal R., and Tjandra I., *Retrieving Information from Data Flow Diagrams*, Department of Computer Science, Concordia University, Montreal, Canada.

D10 Grogono P. and Santas P., *Equality in Object Oriented Languages*, Department of Computer Science, Concordia University, Montreal, Canada and Institute of Scientific Computation, ETH Zurich, Switzerland.

D11 Grogono P. and Gargul M., *A Computational Model for Object Oriented Programming*, Department of Computer Science, Concordia University, Montreal, Canada.

D12 Grogono P. and Gargul M., *A Graph Model for Object Oriented Programming*, Department of Computer Science, Concordia University, Montreal, Canada.

D13 Grogono P. and Gargul M., *Graph Semantics for Object Oriented Programming*, Department of Computer Science, Concordia University, Montreal, Canada.

D14 Grogono P., Taivalsaari A. and Tennenhouse K., *Proposals for Extending the Modelling Facilities of Object Oriented Languages*, Department of Computer Science, Concordia University, Montreal, Canada.

D15 Grogono P., *Issues in the Design of an Object Oriented Programming Language*, Department of Computer Science, Concordia University, Montreal, Canada.

D16 Grogono P., *A Model for Computing with Objects*, Department of Computer Science, Concordia University, Montreal, Canada.

D17 Bouabdalla A., Heydemann M. C., Opatrny J. and Sotteau D., *Embedding Complete Binary Trees into Star Networks*, LIVE institute, Univ. d'Evry-Val-d'Essonne, France, Paris-Sud university, France, and Dept of Computer Sciences, Concordia University, Montreal, Canada.

D18 Paknys R. and Raschkowan L. R., *Moment Method Surface Patch and Wire Grid Accuracy in the Computation of Near Fields*, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada.

D19 Paknys R., *On the Accuracy of the UTD for the Scattering by a Cylinder*, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada.

D20 Davis D., Paknys R., and Kubina S. J., *The Basic Scattering Code Viewer A GUI for the NEC Basic Scattering Code*, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada.

D21 Grogono P. and Cheung B., *A Semantic Browser for Object Oriented Program Development*, Department of Computer Science, Concordia University, Montreal, Canada.

D22 Ounis I., Pasca M., *An Extended Inverted File Approach for Information Retrieval*, Grenoble, France.

D23 Kienle H. M. and Fortier P. J., *Exception-Handling Extension for an Object-oriented DBMS*, University of Stuttgart, Germany and University of Massachusetts Dartmouth, USA.

D24 Nascimento M. A. and Dunham M. H., *A Proposal for Indexing Bitemporal Databases Via Cooperative B+ trees*, Southern Methodist University, Dallas, USA.

D25 Ludwig A., Becker P. and Guntzer U., *Interfacing Online Bibliographic Databases with Z39.50*, University of Tubingen, Germany.

D26 Park C. and Park S., *Alternative Correctness Criteria for Multiversion Concurrency Control and a Locking Protocol via Freezing*, Sogang University, Seoul, Korea.

D27 Woo S., Kim M. H. and Lee Y. J., *Accommodating Logical Logging under Fuzzy Checkpointing in Main Memory Databases*, Department of Computer Science Korea Advanced Institute of Science and Technology, Taejon, Korea.

D28 Cho E. S., Han S. Y., Kim H. J. and Thor M. Y., *A New Data Abstraction Layer Required For OODBMS*, Department of Computer Science and Computer Engineering, Seoul National University, Seoul, Korea.

D29 Ehikioya S. A., *A Formal Specification Strategy for Electronic Commerce*, Department of Computer Science University of Manitoba, Winnipeg, Manitoba, Canada.

D30 Tikekar R. V., *A Generalized Storage Model for Tertiary Storage Based Systems*, Karmanos Cancer Institute, Wayne State University, Detroit, USA.

D31 Kamp V. and Wietek F., *Database System Support for Multidimensional Data Analysis in Environmental Epidemiology*, University of Oldenburg, Germany.

D32 Revesz P. Z. and Li Y., *MLPQ: A Linear Constraint Database System with Aggregate Operators*, Dept. of Computer Science and Engineering, Lincoln, USA.

D33 Khorasani K., *Adaptive Control of Nonlinear Systems Using Output Feedback*, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada.

Appendix B

Oracle ConText's General System Description

The Oracle7 ConText option is a fully integrated text management solution that enables users to process text-based information as quickly and easily as relational data. Oracle context analyses the contents and understands the structure of the English text it reads.

B.1 ConText's General System Description

Context's syntax parser processes each sentence in a document individually. It generates an attribute set representing the syntactic, grammatic, and thematic makeup of the sentences.

Document analysis may continue to the concept Processing Engine. The engine assigns a specific connotative sense to each main theme word. It determines which are the best interpretations of a word based on the concordance of connotations in its generalised structure. The connotation structure is referenced by each word, with each structure containing all possible connotations for the word [41].

B.1.1 ConText's main procedures

1. Sentence Level Processing

   (a) Preparsing Stage

   - Morphological Analysis: Sentence parsing begins when the sentence recogniser extracts and individual sentence from a document and tracks the location of the sentence and each word in the sentence. A language specific routine extracts the base form of each word using suffixes and other clues. This represents the full language lexicon with the smallest number of entries.
   - Grouping words into Syntactic units: Parsing continues by connecting adjacent proper nouns and by checking for any special phrases that transform a group of words into a single syntactic unit.
   - Parsing order for rule sets: The rule set order is based on a weighting mechanism that looks for the least ambiguous words to be parsed first.

   (b) Parsing

   - Parsing Theory and practice: The parser contains both an abstract rule set and a word and phrase specific rule set. The rule sets employed during parsing are mainly function based or English normalising.
   - Lexical Bindings: The word and phrase specific rule set of the parser resides in the lexicon, which is made up of hundreds of possible lexical bindings for each word in the source language.
   - Major Clause Types: The parser contains Subject Verb, or SV, SVO[1], SVC[2], SVA[3], SVOO[4], SVOC[5], and SVOA[6].

[1] SVO stands for Subject Verb Object
[2] SVC stands for Subject Verb Clause
[3] SVA stands for Subject Verb Adjective
[4] SVOO stands for Subject Verb Object Object
[5] SVOC stands for Subject Verb Object Clause
[6] SVO stands for Subject Verb Object Adjective

- Syntactic Descriptions: A Syntactic Description serves as a viewing or storage mechanism. After parsing, the Phrase Structure Grammar analysis is stored.

(c) Rule-Based Parsing

- Normalisation Routine: Normalisation includes expanding or contracting the scope of the possible parts of speech. It also includes several transformation routines that alter the order of words in the sentence.
- Main Syntax Parsing Routines: These routines attempt to prove the correctness of a sentence. The backtracking routines will then analyse the collected information.
- Data Processed
- Syntax Parsing Rules
- Phrase Formation
- Logic Change rules
- Normalisation Rules

2. Concept Processing

(a) Content Analysis

(b) Content Reduction system: Applies two processes. The first process reduces the sentence by removing the least important information. The second process reduces the sentence by extracting the most important information.

(c) Discourse Tracking

- Theme Calculations: The first calculation accumulates the themes through the document on an equal basis. The second calculation finds the pertinence of the thematic information in the discourse. The third calculation type creates an isolated topic list by sentence.
- Theme Types: The theme blocks are document, chapter, section, area, paragraph and sentence.

(d) Connotation Structure

B.1.2 ConText's Support for many languages and formats

Oracle ConText Option is immediately useful to Oracle customers worldwide, because it supports searches of text in many languages, including English, Dutch, German, French, Italian, Spanish and Japanese. It also supports searches of text in most popular formats such as ASCII, HTML, MS Word, and WordPerfect even documents stored outside the database.

Bibliography

[1] Alvarado S., et al, *Argument comprehension and retrieval for editorial text*, Knowledge Based Systems 3 (3), pp. 139-162, 1990.

[2] Andrews K., *The development of a fast conflation algorithm for English*, Dissertation submitted for the Diploma in Computer Science, University of Cambridge, 1971.

[3] *Automatic indexing and classification of the WAIS databases*: http://www.ub2.lu.se/autoclass.html.

[4] Baxendale P. B., *Man made Index for Technical Literature - An Experiment*, IBM Journal of Research and Development, 2:4, pp. 354-361, 1958.

[5] Belkin N., Croft W. B., *Retrieval techniques*, Annual review of information science and technology (ARIST), 22, pp. 109-145, 1987.

[6] Blair D. C. , *Language representation in Information Retrieval*, Elsevier Science publishers, New York, 1990.

[7] Brandow R. , Mitze K., Rau L. F., *Automatic condensation of electronic publications by sentence selection*, Information Processing and management, Vol. 31, No. 5., pp. 675-685, 1995.

[8] Chiaramella Y. et al., *IOTA: a full text information retrieval system*, In proceedings of the ACM conference on research and Development in information retrieval, edited F. Rabitti, Pisa, pp. 207-213, 1987.

[9] De Bra, P., Houben, G-J., & Kornatzky, Y., *Search in the World-Wide Web*, http://www.win.tue.nl/help/doc/demo.ps

[10] Desai, B. C., *An Introduction to Database Systems*, West, St. Paul, MN 1990.

[11] Desai B. C., *Cover page aka Semantic Header*, http://www.cs.concordia.ca/faculty/bcdesai/semantic-header.html, July 1994, revised version, August 1994.

[12] Desai B. C., *The Semantic Header Indexing and Searching on the internet*, Department of Computer Science, Concordia University. Montreal, Canada, February 1995. http://www.cs.concordia.ca/ faculty/bcdesai/cindi-system-1.1.html

[13] DuRoss Liddy E., *Anaphora in natural language processing and information retrieval*, Information Processing and Management, Vol. 26, No. 1, pp. 39-52, 1990.

[14] Earl L. L., *Experiments in Automatic Extracting and Indexing*, Information Storage and Retrieval, 6:4, pp. 313-334, October 1970.

[15] Edmundson H. P., *Problems in Automatic Abstracting*, Communications of the ACM, 7:4, pp. 259-263, April 1964.

[16] Edmundson H. P. and Wyllys R. E., *Automatic Abstracting and Indexing Survey and Recommendations*, Communications of ACM, 4:5, pp. 226-234, May 1961.

[17] Edmundson H. P., *New methods in Automatic Extracting*, University of Maryland, college park, Maryland, Journal of the Association for computing machinery, Vol. 16, No. 2, pp. 264-285, April 1969.

[18] Evans D. A. , *Concept management in text via natural language processing: the CLARIT approach*, In working notes for the AAAI spring symposium on Text-based intelligent systems. Stanford 1990.

[19] Fung R. and Del Favero B. , *Applying Bayesian Networks to Information Retrieval*, Communications of the ACM, Vol 38, No. 3, pp. 42-57, March 1995.

[20] Fletcher, J. 1993., Jumpstation, http://www.stir.ac.uk/jsbin/js

[21] Graham I., *Introduction To HTML and URLs*: http://www.utoronto.ca/webdocs/HTMLdocs/NewHTML/intro.html, Last Update: 24 January 1997.

[22] Hardy D. R., Shwartz M. F., *Customized Information Extraction as a Basis for Resource Discovery*, Department of Computer Science, University of Colorado. March 1994; Revised February 1995.

[23] Jacobs, P.S. and Rau, L.F., *SCISOR: extracting information from online news*, Communications of the ACM, Vol. 33, No. 11, 1990.

[24] Jacqueline W. T. Wong, W. K. Kan, Gilbert Young, *ACTION: Automatic Classification for full-text documents*, ACM Transactions on information systems, pp. 26-41, 1997.

[25] Jing Y. and Croft B. W., *An Association Thesaurus for Information Retrieval*, Department of Computer Science, University of Massachusetts at Amherst, Amherst, MA 01003.

[26] Johnson F. C., Paice C. D., Black W.J, Neal A. P., *The application of linguistic processing to automatic abstract generation*, Journal of Documentation and Text management, 1993.

[27] Katz, W. A., *Introduction to Reference Work*, Vol. 1-2 McGraw-Hill, New York, NY.

[28] Koster, M., *ALIWEB(Archie Like Indexing the WEB)*, http://web.nexor.co.uk/aliweb/doc/aliweb.html

[29] Krovetz R., Croft W. B.. *Lexical ambiguity and information retrieval*, ACM Transactions on information systems, Vol. 10, No. 2, pp. 115-141, April 1992.

[30] Kupiec, J. , Pederson, J., and Chen F., *A trainable document summarizer*, In preceedings of the 18th ACMSGIR Conference, 1995.

[31] Lamport L., *LATEX: A Document Preparation System*, Addison_Wesley, Reading, Massachusetts, second edition, 1994, ISBN 0-201-52983-1.

[32] Lebowitz M., *The use of memory in text processing*, Communications of the ACM, Vol. 33, No. 8, pp. 30-49, 1990.

[33] Lehnert, W. G. and Sundheim, B. 1991. *A Performance Evaluation of Text Analysis Technologies*, AI Magazine 12(3):81-94.

[34] Lewis D. D. , Jones K. , *Natural Language processing for information Retrieval*, Communications of the ACM, Vol 39, pp. 92-101, January 1996.

[35] Lovins J. B., *Development of a Stemming Algorithm*, Mechanical Translation and Computational Linguistics, Vol 11, January 1968.

[36] Luhn, H. P., *The automatic creation of literature abstracts*, IBM Journal of Research and Development, 2, pp. 159-165, 1958.

[37] Maron, M. E. and Kuhns, J. L., *On relevance, probabilistic indexing and information retrieval*, Journal of the ACM, 7, pp. 216-244, 1960.

[38] Mauldin M. 1991. *Retrieval Performance in FERRET: A conceptual Information Retrieval System*, In Proceedings, SGIR 1991. pp. 347-355.

[39] McBryan, Oliver A., *World Wide Web Worm*, http://www.cs.colorado.edu/home/mcbryan/WWWW.html

[40] O'Brien T., *Oracle ConText, Text looms as the next frontier in Information Management*, prepared by Oracle Corporation, April 1996.

[41] Oracle Corporation, *ConText: Introduction to Oracle ConText*, Oracle Corporation, Sept. 1993.

[42] Paice C. D., *Automatic Generation of Literature Abstracts - An Approach Based on the identification of self indicating phrases, in information retrieval research*, R.N. Oddy, S.E. Robertson, C.J. van Rijsbergen and P.W. Williams, editors, Butterworths, London, pp. 172-191, 1981.

[43] Paice C. D., *Constructing Literature Abstracts by Computer: Techniques and Prospects*, Information Processing and Management, 26:1, pp. 171-186, 1990.

[44] Porter, M. F., *An Algorithm For Suffix Stripping*, Program 14 (3), pp. 130-137, July 1980.

[45] Rau, L. F., Jacobs, P. S. and Zernik, U., *Information extraction and text summarization using lingustic knowledge acquisition*, Information processing and management, Vol. 25, No. 4, pp. 419-428, 1989.

[46] Rijsbergen C. J. van, *Information Retrieval*, second edition, Butterworths, pp. 17-22, 1979.

[47] Riloff E. and Hollar L., *Text Database and Information Retrieval*, ACM computing surveys, Vol. 28, No. 1, pp. 133-135, March 1996.

[48] Riloff E. and Lehnert W., *Information Extraction as a basis for High-Precision Text Classification*, ACM Transactions on Information Systems, Vol. 12, No. 3, pp. 296-333, July 1994.

[49] Rush J. E., Salvador R., and Zamora A., *Automatic Abstracting and Indexing-Production of Indicative Abstracts By Application of Contextual Inference and Syntactic Coherence Criteria*, Journal of the ASIS, 22:4,pp. 260-274, July-August 1964.

[50] Salton G. and Lesk M. E., *Computer Evaluation of Indexing and text processing*, Journal of ACM, Vol 25, No. 1, pp. 8-36, 1968.

[51] Salton G., *The SMART Retrieval System*, Prentice-Hall Inc.,4-6, 1971.

[52] Salton G., McGILL M. J., *Introduction to Modern Information Retrieval*, McGraw-Hill Book Company, pp. 87-89, 1983.

[53] Salton G., *Automatic Text Processing: The Transformation, Analysis, and Retrieval of information by Computer*, Addison-Welsey, Reading, MA., 1989.

[54] Salton G., Allen J. , Buckley O. , *Automatic Structuring and Retrieval of Large Text Files*, Department of Computer Science, Cornell University. 1992.

[55] Salton G., Allan J. , Buckley C., and Singhal A. , *Automatic Analysis, Theme Generation, and Summarization of Machine-Readable Texts*, Science, Vol264, pp. 1421-1426, June 1994.

[56] Shayan N., *CINDI: Concordia INdexing and DIscovery system*, Department of Computer Science, Concordia University, Montreal, Canada, 1997.

[57] Smeaton A. F., *Progress in the Application of Natural Language Processing to Information Retrieval tasks*, The Computer Journal, Vol. 35, No. 3, pp. 268-271, 1992.

[58] Stiles, H. F., *The association factor in information retrieval*, Journal of the ACM, 8, pp. 271-279, 1961.

[59] Teufel S. and Moens M., *Sentence extraction as a classification task*, ACL/EACL'97, Intelligent Scalable Text Summarization, Workshop Program, JULY 11, 1997.

[60] Thau, R., *SiteIndex Transducer*,
http://www.ai.mit.edu/tools/site-index.html

[61] Turtle H. R. and Croft, W. B., *Efficient Probabilistic Inference for Text Retrieval*, In Proceedings of RIAO 91. pp. 644-661, 1991.

[62] Computer and Control Abstracts, Produced by INSPEC, No. 10, October 1997.

[63] http://www.oracle.com.sg/products/oracle7/oracle7.3/html/conTxtDS.html.

[64] http://www.oracle.com.sg/products/oracle7/oracle7.3/html/context_seybold.html.

[65] *Experimental Search Engine Meta-Index*,
http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Demo/metaindex.html

[66] Library of Congress Subject Headings, September 1996.

[67] http://www.acm.org/class/1998/ccs98.txt.

[68] Search WWW document full text,
http://rbse.jsc.nasa.gov/eichmann/urlsearch.html

[69] WebCrawler,
http://www.biotech.washington.edu/WebCrawler/WebQuery.html

[70] World Wide Web Catalog,
http://cuiwww.unige.ch/cgi-bin/w3catalog

[71] http://web.soi.city.ac.uk/research/cisr/okapi/stem.html

[72] http://www.qpat.com/info/help/stemhelp.html