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.

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.

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

EXTRACTION OF SEMANTIC HEADER FROM RTF DOCUMENTS

ABDELBASET ALI

A Major Report
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec

August, 1999

Abstract

Extraction of Semantic Header from RTF Documents

Abdelbaset Ali

The problem of indexing and retrieval of electronic information resources becomes more critical as the amount of information and the number of Internet users continues to grow. The Semantic Header, proposed by Desai [3], is a portion of each document that contains the meta-information for each publicly accessible resource on the Internet. The Semantic Header for document-like Internet resources is a powerful means of helping users locate documents 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.

In this project which is a part of the ASHG system (Automatic Semantic Header Generator), we present a model for type-specific, information extraction that automatically extracts the meta-information from RTF (Rich Text Format) documents, and stores it in a Semantic Header which will be used as an index for the document. This shall provide 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.

Acknowledgements

I would like to thank my supervisor, Dr. Desai, for his guidance and encouragement throughout this project. I would like to express my deepest gratitude to him for having made my project work a pleasant and extremely educational experience. I am forever indebted to you Dr Desai, and I am hopeful that we will meet and work again sometime in the future.

To all my friends and colleagues in the Department of Computer Science, thank you for your encouragement, thoughtful discussions, and help.

I would like to thank my friends for being there for me in times of need.

Finally, I would like to thank my wife for her patience and encouragement.

To the memory of my parents

Contents

List of Figures viii

List of Tables ix

1. Introduction 1

1.1 The Discovery Problem 1

1.2 CINDI's approach 1

1.2.1 Meta-Information Indexing 2

1.3 Organization of the report 2

2. The CINDI System 3

2.1 Overview of CINDI 3

2.1.1 The Semantic Header of CINDI 4

2.1.2 Semantic Header's Mark-up Schema 6

2.1.3 Database System of the Semantic Header 7

2.1.4 CINDI's Search System 8

3. Rich Text Format 9

3.1 Rich Text Format (RTF) Specification 9

3.1.1 RTF Syntax 9

3.1.2 Contents of an RTF file 10

3.1.2.1 Document Area 10

4. Automatic Semantic Header Generator (ASHG) 13

4.1 Introduction 13
4.3 Document Type Recognition 14
4.4 The Thesaurus for ASHG 16
4.4.1 The subject Hierarchies 16
4.5 ASHG's Document Subject Headings Classification scheme 17
4.5.1 The Algorithm 17
4.6 Applying ASHG's Extractors 18
4.6.1 RTF_extractor 18
4.6.2 Generating an implicit list of keywords 21
4.7 Semantic Header Validation 25
5. ASHG's Experiments 26
5.1 Experiments 26
5.2 Sample Results 28
6. Conclusion and Future Work 37
6.1 Conclusion 37
6.2 Future Work 38
References 42

List of Figures

Figure 1: ASHG's steps 16

Figure 2: Subject Extraction 19

Figure 3: RTF extractor.....21

Figure 4: ASHG extraction for RTF document.....24

Figure 5: CINDI' Semantic Header Graphical interface (a) 33

Figure 6: CINDI' Semantic Header Graphical interface (b) 34

Figure 7: CINDI' Semantic Header Graphical interface (c).....35

List of Tables

Table 1: Definitions of Control words (a) **12**

Table 2: Definitions of Control words (b) **12**

Table 3: Summary of ASHG's RTF test against the authors and INSPEC's results **28**

Chapter 1

1. Introduction

1.1 The Discovery Problem

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, local and national directories of various types, and private information services. Rapid growth in data volume, user base and data diversity renders Internet-accessible information increasingly difficult to locate efficiently. Browsing a hierarchy containing millions of directories is infeasible, particularly given the erratic organization that results when many different people create the data. Instead, there is a need for an automated search and for a system that allows easy "search for and access to" resources available on the Internet. For this automated search to help users in locating relevant information require indexing the available information. Thus, secondary information called meta-information must be extracted and used as an index to the available primary resource. In turn, building this index requires information extraction methods tailored to each specific environment. To this end, the semantic of the files in which the primary resource is stored will be exploited to extract and summarize the relevant information. 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 envisaged that regional and/or specialized databases would be created to maintain archives of Semantic Header. These databases could be searched by the cooperation of distributed expert systems to help users locate pertinent documents. Such a system is currently under development at Concordia University and is called Concordia Indexing and Discovery systems, or CINDI.

1.2 CINDI's approach

CINDI (Concordia Indexing and Discovery system), a system under development at Concordia University, provides a mechanism to register, search and manage the meta-information, with the help of easy to use graphical user interface. The meta-information, which is described in chapter 2, is the Semantic Header, that is stored in the CINDI system. CINDI has been designed to avoid problems caused by differences in semantics and representation, and incomplete or incorrect data cataloging. This meta-information could either be entered by the primary resource provider or assisted in this by the Automatic Semantic Header Generator (ASHG). ASHG is a software that generates some meta-information of the submitted document, and assists the user in this process. This major report introduces ASHG for RTF documents, which aims at saving time for the

primary resource provider 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. The provider helps in this process by verifying and correcting the Semantic Header entry.

1.2.1 Meta-Information Indexing

Constructed meta-information could support queries based on content as well as traditional queries based on items such as title, author, subject, etc. This means that actual sources could be separated from their meta-information, thereby simplifying the storage of the resources. Professional catalogers have found the need for the mentioned elements in indexing applications. The dependence on titles as the most commonly used search criterion suggests that they must indicate the contents of the document. Furthermore, the author or the cataloger has to add annotations, keywords, or key phrases to indicate its actual content. Including reviewers' opinions can indicate the accuracy or quality of a document. However, such opinions are rarely accessible to the traditional cataloger. Another feature of importance is the presence of an accurate abstract. An abstract provides a summary of the material, and thus is more indicative of the contents than the title and keywords supplied by the author [14].

1.3 Organization of the report

This report is organized as follows. In Chapter 2, the CINDI system is introduced. Chapter 3 describes Rich Text Format (RTF). Chapter 4 describes the Automatic Semantic Header Generator (ASHG). This chapter covers the type recognition and the extractors, and a description of the Thesaurus used at the beginning of this chapter. In chapter 5, we test and compare the classification of our generated index with those produced by cataloguers. Lastly, in chapter 6, we give our conclusion and discuss future work.

Chapter 2

2. The CINDI System

2.1 Overview of CINDI

The trend in most research institutes, universities and business organization of interconnecting their computing facilities using a digital network has become the accepted method of sharing resources. Such networks, in turn, are interconnected allowing information to be exchanged across networks by using a common interchange protocol (TCP/IP). The number of such interconnected networks (Internet) continues to grow and with the emergence of powerful workstation-based servers connected to these networks, it is possible to support local as well as remote search and retrieval of information stored on any component of the interconnection .

There is a need for the development of a system, which allows easy search for and access to resources available on the Internet. It has been observed that distributed information systems, even though under control of a single administrative unit, create multiple problems typically caused by differences in semantics and representation, and incomplete and incorrect data dictionaries. The building of a standard index structure and a bibliographic system using standardized control definitions and terms can solve these problems and lead to a fast, efficient and easy access to the documents. Such definitions could be built into the knowledge base of expert system based index entry and search interfaces. The purpose of indices and secondary information is to inventory the primary information and allow easy access to it. Preparing a secondary information requires finding the primary source, identifying it as to its author, title subject, abstract, keywords, etc. Since an index is to be used by many users, it has to be accurate, easy to use (usage via author, title, subject, etc.) and properly classified.

Attempts to provide easy searching of relevant documents has lead to a number of systems including WAIS, and more recently a number of Spiders, Worms and other creepy crawlers [1], [6], [10], [12], [16], [15], [17], [18]. However the problem with many of these indices is that their selectivity of documents is often poor [4]. The chances of getting correct documents and missing relevant information because of poor choice of search terms are large. In addition, the user is required to access the actual resource, based just on the title and author information as provided through a library catalog, and decide whether the resource meets the needs. These problems are addressed by CINDI using an appropriate index entry called Semantic Header which includes title, abstract, keywords, and subject; and providing a mechanism to register, manage and search the bibliography.

The overall CINDI system uses knowledge bases and expert sub-systems to help the user in the register and search process. The expert system will help the user in entering

those attributes required for indexing or updating a Semantic Header. CINDI standardizes 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 [3].

2.1.1 The Semantic Header of CINDI

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 authors (70%), subject and sub-subject (50%) [9], CINDI requires the entry for these elements in the Semantic Header. Similarly, since the abstract and annotations are relevant in deciding whether or not a resource is useful, they are also included. [14,4]. The description 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-subject 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 same as those used in the 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 identifier for the document. Example of identifier 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 the document will expire.

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

Classification
The classification of the document such as 'legal', 'security', or other types. For each, the nature of classification is specified.

Coverage
It indicates either the target audience of the document or the cultural and temporal aspects of the document's content.

System Requirements
Being an electronic document certain system requirements for it to be displayed or used are necessitated. The components involved are the hardware, the software, or the network and the minimum needs for each of these.

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

Source and Reference
The Source indicates the documents being referenced or that was 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 heart of any bibliography or indexing system is the record that is kept for each item that is being indexed. The syntax of the semantic header is the HTML markup language, which is based on the SGML markup language. The following entries will form the meta-information of the primary resource, which will be stored in the Semantic Header database. Once this index is filled by the provider of the resource or by the proposed ASGH system, it will be registered as bibliographical information about the resource.

<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 name, organization, address, etc. of each person/institute responsible for the information resource: at the least, the name of the organization 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>

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

<Classification>OPTIONAL: nature security level of the resource</Classification>

<URL>A list of locations (URL) Unique Universal Resource Locator-Call No for this resource: at least one required</URL>

<Abstract>OPTIONAL but recommended</Abstract>

<Annotation>OPTIONAL</Annotation>

<SysReq>OPTIONAL: list of requirements in hardware and software

<Hardware>OPTIONAL: list of hardware required</Hardware>

<Software>OPTIONAL: list of software required</Software>

</SysReq>

<size>size of the resource in 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>

2.1.3 Database System of the Semantic Header

The index entry system provides a graphical interface to facilitate the provider of a resource to register the bibliographic information about the resource. Once the information is correctly entered, the provider can decide to register the Semantic Header entry in the database. The index entries registered by a provider of a resource are stored in a distributed database system (SHDDB). The underlying database may be considered to be a monolithic system from the point of view of the users of the 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 [2].

The Semantic Header information entered by the provider of the resource using a graphical interface is relayed by client process from the user's workstation 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.

2.1.4 CINDI's Search System

The search system also uses a graphical interface and a client process allow a provider to enter search requirements consisting of sub-string and synonyms. The search interface providing the precise statement of a user query by allowing complex predicates based on search items such as author, subject, keywords, dates, and words appearing in the abstract. 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 then 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 is 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 [4].

Chapter 3

3. Rich Text Format

3.1 Rich Text Format (RTF) Specification

RTF (Rich Text Format) allows for the exchange of text files between different word processors and operating systems. For example, a file created using Microsoft Word 97 in Windows 95 can be saved and sent as an RTF file to someone using Wordperfect 6.0 on Windows 3.1. This RTF file will have a ".rtf" file name suffix.

Control words and symbols serving as "common denominator" formatting commands are defined by RTF specifications using the PC-8, Macintosh, ANSI, and IBM character sets. An RTF writer processes files saved in the Rich Text Format and convert's the word processor's markup to the RTF language. The RTF reader processes and converts the RTF language so that it can be displayed on a different word processor. In this way, the Rich Text Format (RTF) Specifications [13] serves as a 'bridge' that allows for the interchange of text and graphics between different operating systems, operating environments, and output devices.

The actual software that turns a formatted file into an RTF file is called a writer. It writes a new file containing the text and associated RTF groups. This RTF file is then translated into a formatted file by a reader [13].

3.1.1 RTF Syntax

An RTF file is comprised of control words, control symbols, unformatted text, and groups [13]. A control word takes the following form:

\ LetterSequence<Delimiter>

Note that each control word is preceded by a backslash.

The LetterSequence is made up of lowercase alphabetic characters.

A control symbol consists of a single, non-alphabetic character preceded by a backslash. For example, \~ represents a non-breaking space. Control symbols take no delimiters [13].

A group consists of text and control words or control symbols enclosed in braces ({ }). The start of the group is marked by an opening brace ({) and the end of the group by a closing brace (}). Each group specifies the text affected by the group and the different attributes of that text. The RTF file can also include groups for fonts, styles, screen color, pictures, footnotes, annotations, headers and footers, summary information, fields, and bookmarks, as well as document-, section-, paragraph-, and character-formatting properties. If the font, file, style, screen-color, revision mark, summary-information groups and document-formatting properties are included, they must precede the first plain text character in the document. These groups form the RTF file header [13].

Some control words have only two states. For example, bold, which is either turned on or turned off. When such a control word has no parameter or has a nonzero parameter, it is assumed that the control word turns on the property. When such a control word has a parameter of 0 (zero), it is assumed that the control word turns off the property. For example, **\b** turns on bold, whereas **\b0** turns off bold [13].

Within a document, the beginning of a collection of related text that could appear at another position, or destination, is distinguished by certain control words referred to as destinations. Braces must enclose destination control words and any text following them. The control words, control symbols, and braces constitute control information. All other characters in the file are plain text. As previously mentioned, the backslash (and braces ({ }) have special meaning in RTF. To use these characters as text, precede them with a backslash, as in \\, \{ and \} [13].

3.1.2 Contents of an RTF file

An RTF file has the following syntax:

<File> '{' <header> <document>'}'

It is important to include here an explanation of the document area because RTF_extractor exploits the use of mark-up fields to extract the meta-information. It extracts the title, explicitly stated keywords, subject, language, author(s), dates (Created, Expiry), and the abstract from the RTF document. This will be described in another chapter.

3.1.2.1 Document Area

The document area has the following syntax.
<document> <info>?<docfmt>*<section>+

Information group

The information group, which contains information about the document, is introduced by the **\info** control word. This can include the title, author, keywords, comments, and other information specific to the file [13]. This group has the following syntax:

<info> '{' <title>? & <subject>? & <author>? & <operator>? & <keywords>? & <comment>? & \ ***version***? & <doccomm>? & \ ***vern***? & <creatim>? & <revtim>? & <printim>? & <buptim>? & \ ***edmins***? & \ ***nofpages***? & \ ***nofwords***? & \ ***id***? '}'

<title> '{' \ **title** #PCDATA '}'

<subject> '{' \ **subject** #PCDATA '}'

<author> '{' \ **author** #PCDATA '}'

<operator> '{' \ **operator** #PCDATA '}'

<keywords> '{' \ **keywords** #PCDATA '}'

<comment> '{' \ **comment** #PCDATA '}'

<doccomm> '{' \ **doccomm** #PCDATA '}'

<creatim> '{' \ **creatim** <time> '}'

<revtim> '{' \ **revtim** <time> '}'

<printim> '{' \ **printim** <time> '}'

<buptim> '{' \ **buptim** <time> '}'

<time> \ **yr**? \ **mo**? \ **dy**? \ **hr**? \ **min**? \ **sec**?

#PCDATA : Text (without control words)

Some applications, such as Word, ask a user to type this information when saving the document in its native format. If the document is then saved as an RTF file or translated into RTF, the RTF writer specifies this information using the following control words. These control words are destinations and both the control words and the text should be enclosed in braces ({}) [13].

Control word	Meaning
\ title	Title of the document.
\ subject	Subject of the document.
\ author	Author of the document.
\ operator	Person who last made changes to the document.
\ keywords	Selected keywords for the document.
\ comment	Comments; text is ignored.
\ version*N*	The version number of the document.
\ doccomm	Comments displayed in Word's Edit Summary Info dialog box.

Table 1: Definitions of Control words (a)

The RTF writer can automatically enter other control words, including the following:

Control word	Meaning
\ vern*N*	The internal version number
\ creatim	The creation time
\ revtim	The revision time
\ printim	The last print time
\ buptim	The backup time
\ edmins*N*	The total editing time (in minutes)
\ yr*N*	The year
\ mo*N*	The month
\ dy*N*	The day
\ hr*N*	The hour
\ min*N*	The minute
\ sec*N*	The seconds
\ nofpages*N*	The number of pages
\ nofwords*N*	The number of words
\ nofchars*N*	The number of characters
\ id*N*	The internal ID number

Table 2: Definitions of Control words (b)

Entries without the N parameter have the **\ yr \ mo \ dy \ hr \ min \ sec** controls [13]. An example of an information group follows:

{\info{\title The Semantic Header and Indexing and searching on the Internet}{\author Bipin C. Desai}{\keywords Bibliographic record, Content description, Indexing, Index aid, Database system, Expert system, Searching, URC, URL }}

Chapter 4

4. Automatic Semantic Header Generator (ASHG)

4.1 Introduction

In this chapter, the Automatic Semantic Header Generator (ASHG) of the CINDI for RTF documents with ASHG is presented. ASHG saves time for the document's author by generating an initial set of subject classification and a number of components of the Semantic Header for the document. ASHG measures both the frequency of occurrence and positional weight of keywords found in the document. By matching those keywords with the controlled terms found in the controlled term subject association a list of subject headings is then generated [7].

The selection of important terms may be based either upon position (the term's location in the document), semantic, or pragmatic (a system which would consider proper names as highly significant). Another criterion used for selecting important terms is Statistical term weight. 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 [11,5].

Meta-information from HTML, Latex, Text, RTF, and Unknown documents is extracted by the ASHG and is stored in a Semantic Header. This meta-information may include such fields as the document's title, abstract, keywords, dates, author, author's information, size and version. The significance of these is measured by ASHG using frequency occurrence and positional schemes. A base form for each word is generated by Word stemming and then matched to the controlled terms found in the controlled term subject association [7]. If a match is found the subject headings associated with the controlled terms are extracted and ranked accordingly. The following describes the major steps of ASHG.

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

- ***File Type Validation:*** The user validates the file type recognized by ASHG.

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

- ***ASHG's Document Classification:*** A classification of subject headings is assigned to the document. It involves:

    1. ***Word stemming:*** A stemming process is applied by the system to map the words found in the extracted fields onto a base root word.

    2. ***A Look-up into the Controlled Term Subject dictionary.***

- ***Semantic Header Validation:*** The generated Semantic Header is presented to the user to validate, correct, and register using the GUI [21].

4.3 Document Type Recognition

Name conventions are used by the system to assist in recognition of document type. If this does not work, the system will then examine the contents. If the document type remains unrecognized, the user is informed by the system, and is asked to either choose a document type or generate the Semantic Header manually. The two steps used in the document type recognition process follow:

- Naming conventions and heuristics.

- Examining file contents in determining the file types.

Upon submitting, the document file 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, text or RTF then the function ***user_verify*** is called. If the naming convention fails in recognizing 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 == .rtf or .RTF) then
{
  The file is a rtf 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 recognized, ASHG applies the extractor corresponding to the type confirmed by the user. Otherwise, he should choose a type and then apply ASHG.

In the function ***bycontent***, the semantics of the HTML, Latex and RTF content is exploited when attempting to recognize 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 if (the file contents match the rtf semantics, such as the
  existence of the \rtf tag) then
{
  the file is of rtf type
  call function user_verify
}
else
{
  Unrecognized file type, The user should select a type.
}
```

If the file type is not HTML, Latex, Text, or RTF, it remains unrecognized, and ASHG extracts the size of the file and the date of creation.

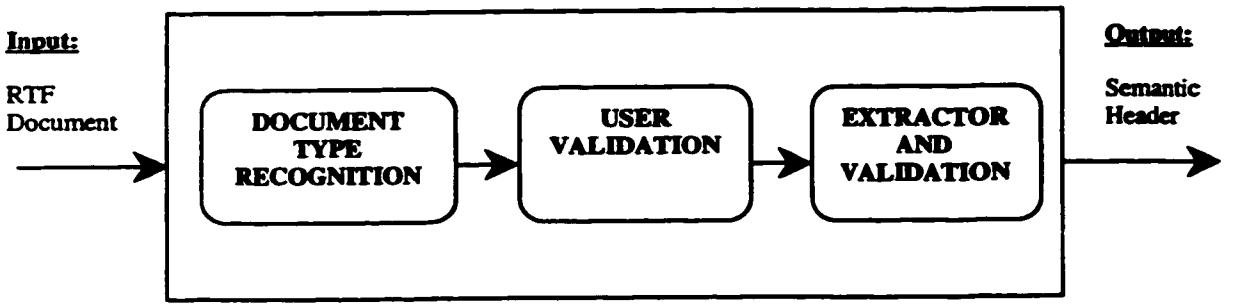

Figure 1: ASHG's steps

4.4 The Thesaurus for ASHG

The ASHG's Thesaurus is composed of a three level subject hierarchy 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 [7].

4.4.1 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 have been derived. The CINDI system focuses on the standardization 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.

4.5 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 by themselves enough to convey the ideas or subjects of the document. Since the author tries to convey or to summarize 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 [7].

4.5.1 The Algorithm

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:-

- 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.

- 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 headings (*list of level 0*, *level 1* and *level 2*), and adds the subject headings to their corresponding list of possible subject headings.

- 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 occurring in both title and abstract, for instance, gets a weight of seven. The matched terms' weights are passed to their subject headings.

- The system extracts *Level_2*, *Level_1* and *Level_0* subject headings having the highest weights from the three lists of possible subject headings.

- After building the three lists for the three level subject headings, the system:

    1. Selects the subjects using the bottom-up scheme.

    2. Having selected the highest weighted level_2 subject headings, the system derives their level_1 parent subject headings.

3. 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.

4. The system uses the same procedure in selecting level_0 subject headings.

4.6 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, text and RTF syntax documents to extract the document's meta-information. ASHG's ***RTF_extractor,*** are applied to ***RTF*** type documents and to unrecognized type documents 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 ***HTML, Latex*** and ***RTF*** 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 ***RTF*** document, The ***RTF_extractor*** extracts words that are between keywords control word braces in **{\keywords...}**. 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.

4.6.1 RTF_extractor

After identifying the document's type, the corresponding extractor is applied to the document, thus aiming to retrieve the corresponding meta-information to build its Semantic Header.

The ***RTF_extractor*** extracts the title, explicitly stated keywords, language (English), author(s), dates (Created, Expiry), size of the file, and the abstract from a *RTF* 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.

1) ***Extracting the title from RTF document:*** The title could be extracted from what is between title control word braces in **{\title...}**. For example, if the RTF document contains {\title *Automatic Semantic Header Generator*}, RTF_extractor extracts *Automatic Semantic Header Generator* as the document's title. If this fails, it then extracts the first sentence.

2) ***Extracting the subject from RTF document:*** The subject could be extracted from what is between control word braces in **{\subject...}**, or the RTF_extractor uses the resulting list of significant words generated from the subsection: 4.6.2 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. The subject will be selected only after the keywords from the documents. And these keywords will lead the system in identifying the right subject by using the Keyword-Subject database.

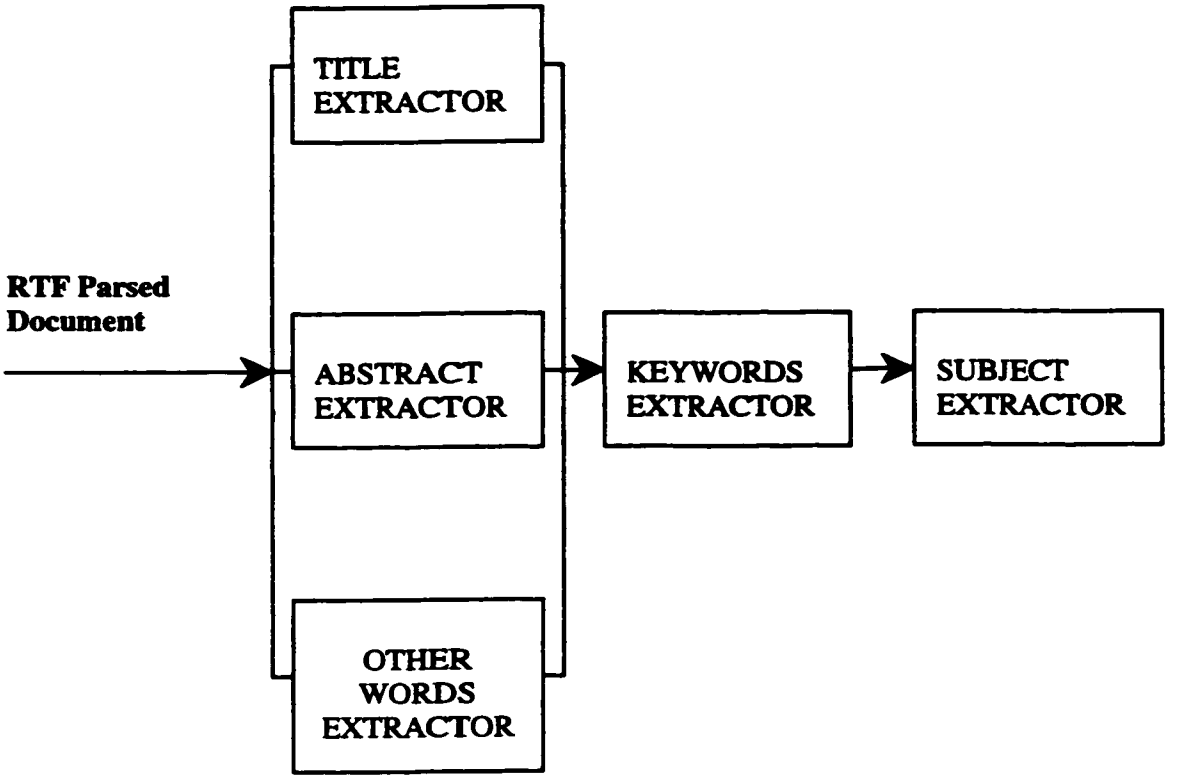

Figure 2: Subject Extraction

3) ***Extracting the language from RTF document:*** The language could be extracted from what is after **\deflang** control word (default is English).

4) ***Extracting the author(s) RTF document:*** The author name is extracted from what is between author control word braces in **{\author...}**. For instance, if the RTF document contains {\author *Abdelbaset Ali*}, the *RTF_extractor* extracts *Abdelbaset Ali* as the author of the document. Extraction can also be made by looking for the patterns: edited by, written by, revised by, author, when selecting the author and what follows it which will then be assumed to be the author. The following information could also be extracted:

- ❖ ***The e_mail:*** will be extracted by looking for the "@" symbol which is the worldwide usable symbol for an e_mail.

- ❖ ***The phone:*** will be extracted by looking for the pattern phone or tel.

- ❖ ***The fax:*** will be extracted by looking for the pattern fax.

5) ***Extracting the abstract from RTF document:*** The abstract could be extracted from what is between control word braces in **{\abstract.....}**. If it fails, it then 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, 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 *RTF_extractor* extracts the first paragraph after removing the RTF tags and applies the automatic abstracting method, described above, to this paragraph.

6) ***Extracting explicitly stated keywords from RTF document:*** The keywords are extracted from what is between keywords control word braces in **{\keywords...}**. For example, if the RTF document contains control word {\keywords type , require, object , define , act , abstract , issue}, the RTF_extractor extracts these as the documents's keywords. If it fails then the pattern keywords are looked for, and what follows are assumed to be the keywords of the document.

7) ***Extracting the created date from RTF document:*** The created date could be extracted from what is after **\creatim** control word. For example, if {\creatim\yr1999\mo8\dy10\hr12\min9} was found in the RTF document, *RTF_extractor* extracts *10/08/99* as the document's date. If no \creatim command is found in the RTF document, *RTF_extractor* 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.

8) ***Extracting other tagged words from RTF document:*** The *RTF_extractor* extracts a list of tagged words. For example, if the RTF document contains the meta tags \b Computer \b0, the RTF_extractor includes *Computer* in the list of other tagged 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 will be described in subsection 4.6.2.

9) ***Extracting the version from RTF document:*** The version could be extracted from what is after **\version** control word.

10) ***Extracting the size of the RTF document:*** using the stat unix command, the size of the file can be extracted.

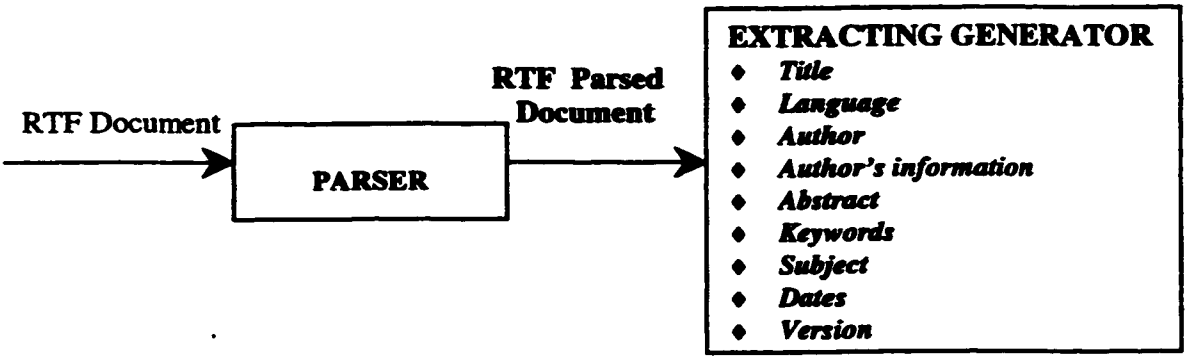

Figure 3: RTF extractor

4.6.2 Generating an implicit list of keywords

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

In case keywords are 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 are 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, this algoritm being the same used for the extraction implicit keywords from HTML, Latex documents [7]. In cases where the keywords are not found in the document, and the words used in the classification scheme:-

- Extract the title, abstract and other tagged fields.

- English Noise words constitute usually around 30 to 50 percent of a document. The Information Retrieval community calls them the ***Stop List*** [7]. These words are dropped from the extracted fields.

- The remaining words are sent to the stemming process. This process will remove the words' suffixes and prefixes. The aim of the stemming process is to generate a base word class, which includes all the forms that could be generated from it.

- 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 weights for the less important terms [19]. 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:

    1. If the word appears in the explicitly stated keywords, it is assigned a weight of five. Since authors explicitly state the keywords to convey some important terms that their document covers, it is assigned the highest weight.

    2. 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 since they convey the idea of the article more than any words found in other locations [20]. If the word appears in the abstract, it is assigned a weight of four.

    3. If the word appears in the title, it is assigned a weight of three.

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

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

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

    2. A class weight of three defines the TITLE class. This class contains all the terms found only in the title field.

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

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

5. A class weight of six includes all the terms found in both the abstract field and the other words fields.

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

7. A class weight of eight contains all the terms found in keyword and title fields.

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

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

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

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

12. 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 high class weights, we are interested in extracting more terms than in lower class weights. Thus, we tend to extract more terms from the high weight 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 weight 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 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 example, 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 weight 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.

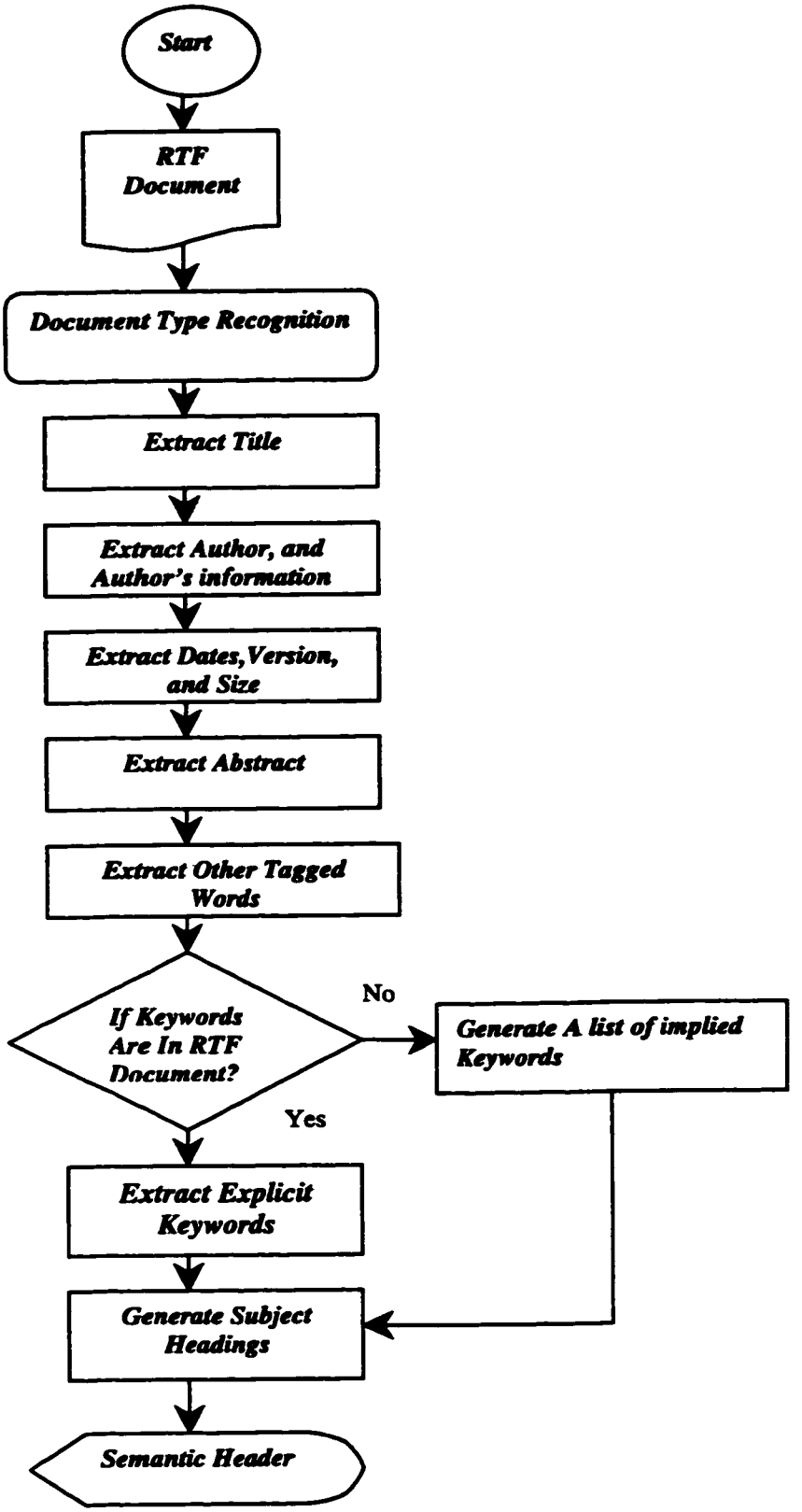

Figure 4: ASHG extraction for RTF document

4.7 Semantic Header Validation

Once the process of extracting the meta-information or secondary information is completed, 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 5

5. ASHG's Experiments

In this chapter, we illustrate how the ASHG system extracts the meta-information from the RTF documents, and demonstrate ASHG's automatic subject heading 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.

5.1 Experiments

After applying the ASHG on a set of RTF documents, the titles of these documents can be viewed in the appendix. 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.

The experiments were conducted on twenty-five documents to test the accuracy of the generated index and the subject heading classification results. These documents deal 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 that included some words that are insignificant. These insignificant words in turn lead to the diversion in subject classification.

Based on the consultation with the papers' authors on the ASHG's subject classification results done by Haddad [7]. 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 that 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. 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. We have compared the result for HTML_extractor for HTML documents, Latex_extractor for Latex documnets, and Text_extractor for text documents with the result for the same document in RTF format which extracted by RTF_extractor.

RTF Document	Number of Subject Headings generated by ASHG (RTF_extractor)	Author's Opinion			Accuracy	OK/Good's Accuracy
		Good	OK/Not Sure	Poor	A: Author I: INSPEC	
Doc1	10	4	4	2	40% (A)	80%
Doc2	6	0	2	4	0 (A)	33.33%
Doc3	4	0	2	2	0 (A)	50%
Doc4	4	0	2	2	0 (A) 25% (I)	50%
Doc5	4	0	4	0	0 (A)	100%
Doc6	5	1	1	3	20% (A)	40%
Doc7	5	0	5	0	0 (A) 20% (I)	100%
Doc8	6	2	3	1	33.33 (A)	83.33%
Doc9	14				14.28% (I)	
Doc10	4				25% (I)	
Doc11	5	1	4	0	20% (A) 20% (I)	100%
Doc12	4	1	2	1	25% (A)	75%
Doc13	3	1	1	1	33.33 (A) 66.66 (I)	66.66%
Doc14	3	1	0	2	50% (A) 50% (I)	50%
Doc15	5	1	0	4	20% (A)	20%

RTF Document	Number of Subject Headings generated by ASHG (RTF_extractor)	Author's Opinion			Accuracy	OK/Good's Accuracy
		Good	OK/Not Sure	Poor	A: Author I: INSPEC	
Doc16	3	0	0	3	0 (A) 33.33 (I)	0
Doc17	7				28.57% (I)	
Doc18	7				14.28% (I)	
Doc19	4				0 (I)	
Doc20	4				25% (I)	
Doc21	3				0 (I)	
Doc22	3				66.66% (I)	
Doc23	26				50% (I)	
Doc24	5				0 (I)	
Doc25	4				25% (I)	
Averages	—	—	—	—	28.98%	60.59%

Table 3: Summary of ASHG's RTF test results against the authors and INSPEC's results

5.2 Sample Results

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

Result 1

<semhdrB>
<useridB> <useridE>
<passwordB> <passwordE>
<titleB> Designing a Class Library <titleE>
<alttitleB> <alttitleE>
<subjectB>
<generalB> Computer Science <generalE>
<sublevel1B> Programming languages <sublevel1E>
<sublevel2B> Computer program language <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Theory of computation by abstract devices <sublevel1E>
<sublevel2B> Complexity measures and classes <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Theory of computation by abstract devices <sublevel1E>
<sublevel2B> Complexity measures and classes hierarchies <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Theory of computation by abstract devices <sublevel1E>

<sublevel2B> Complexity measures and classes reducibility and completeness <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Theory of computation by abstract devices <sublevel1E>
<sublevel2B> Relations among complexity classes <sublevel2E>
<subjectE>
<languageB> English <languageE>
<char-setB> <char-setE>
<authorB>
<aroleB> Author <aroleE>
<anameB> Peter Grogono <anameE>
<aorgB> <aorgE>
<aaddressB> Department of Computer Science, Concordia University de Maisonneuve Blvd., Montreal, Quebec Canada H3G 1M8 <aaddressE>
<aphoneB> 848-3000 <aphoneE>
<afaxB> 848-2830 <afaxE>
<aemailB> grogono@concour.cs.concordia.ca <aemailE>
<authorE>
<keywordB> library , class , programmer , program , problem , presentation , main , lay , internal , environ , do , develop , dee , complex , collect , call , built , browse , advantage , abstract <keywordE>
<identifierB>
<domain3B> FTP <domain3E>
<value3B> <value3E>
<identifierE>
<datesB>
<createdB> 1999/6/14 <createdE>
<expiryB> 1999/6/14 <expiryE>
<datesE>
<versionB> 1 <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> 4503 <sizeE>
<genreE>
<source-referenceB>
<relationB> <relationE>
<domain-identifierB> <domain-identifierE>
<source-referenceE>
<costB> <costE>
<abstractB>
One of the functions of a class library is to provide a collection of data structures. We argue that if a class library is to be useful, it must do more than this: it must provide useful abstractions and it must allow choices of representation to be deferred until late in program development. We have designed a language and a development environment called Dee. The class library of Dee is built in layers and has a complex internal structure. A sophisticated class browser conceals the complexity of the class library from

programmers. We introduce the salient features of Dee, describe the structure of its class library, explain the advantages of this structure, and discuss the remaining problems.
<abstractE>
<annotationB>
<annotationE>
<semhdrE>
<EOF>

Result 2

<semhdrB>
<useridB> <useridE>
<passwordB> <passwordE>
<titleB> Issues in the Design of an Object Oriented Programming <titleE>
<alttitleB> <alttitleE>
<subjectB>
<generalB> Computer Science <generalE>
<sublevel1B> Programming languages <sublevel1E>
<sublevel2B> Abstract data types in language constructs and features <sublevel2E>
<generalB> abstract <generalE>
<sublevel1B> theory of computation by abstract devices <sublevel1E>
<sublevel2B> abstract data types in language constructs and features <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Programming languages <sublevel1E>
<sublevel2B> Classes and objects in language constructs and features <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Theory of computation by abstract devices <sublevel1E>
<sublevel2B> Complexity measures and classes <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Theory of computation by abstract devices <sublevel1E>
<sublevel2B> Complexity measures and classes hierarchies <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Theory of computation by abstract devices <sublevel1E>
<sublevel2B> Complexity measures and classes reducibility and completeness <sublevel2E>
<generalB> Computer Science <generalE>
<sublevel1B> Theory of computation by abstract devices <sublevel1E>
<sublevel2B> Relations among complexity classes <sublevel2E>
<subjectE>
<languageB> English <languageE>
<char-setB> <char-setE>
<authorB>
<aroleB> Author <aroleE>
<anameB> Peter Grogono <anameE>
<aorgB> <aorgE>
<aaddressB> Department of Computer Science, Concordia University de Maisonneuve Blvd. West, Montr
eal, Quebec Canada H3G 1M8 <aaddressE>
<aphoneB> 848-3000 <aphoneE>
<afaxB> 848-2830 <afaxE>
<aemailB> grogono@concour.cs.concordia.ca <aemailE>
<authorE>
<keywordB> type , require , object , define , act , abstract , issue <keywordE>
<identifierB>
<domain3B> FTP <domain3E>
<value3B> <value3E>
<identifierE>

<datesB>
<createdB> 1999/6/4 <createdE>
<expiryB> 1999/6/4 <expiryE>
<datesE>
<versionB> 1 <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> 73741 <sizeE>
<genreE>
<source-referenceB>
<relationB> <relationE>
<domain-identifierB> <domain-identifierE>
<source-referenceE>
<costB> <costE>
<abstractB>
The object oriented paradigm, which advocates bottom-up program development, appears at first sight to run counter to the classical, top-down approach of structured programming. The deep requirement of structured programming, however, is that programming should be based on well-defined abstractions with clear meaning rather than on incidental characteristics of computing machinery. This requirement can be met by object oriented programming and, in fact, object oriented programs may have better structure than programs obtained by functional decomposition.

The definitions of the basic components of object oriented programming, object, class, and inheritance, are still sufficiently fluid to provide many choices for the designer of an object oriented language. Full support of objects in a typed language requires a number of features, including classes, inheritance, genericity, renaming, and redefinition. Although each of these features is simple in itself, interactions between features can become complex. For example, renaming and redefinition may interact in unexpected ways. In this paper, we show that appropriate design choices can yield a language that fulfills the promise of object oriented programming without sacrificing the requirements of structured programming.
<abstractE>
<annotationB>
<annotationE>
<semhdrE>
<EOF>

Example :

The following example shows CINDI Semantic Header Graphical interface, We have used RTF document to extract meta-information. The generated Semantic header is written into a file where it is read and displayed into the CINDI's GUI. The ASHG's result will be used as a starting point by the author, and he/she has the opportunity to correct the errors and include fields of the Semantic Header not given before registering it.

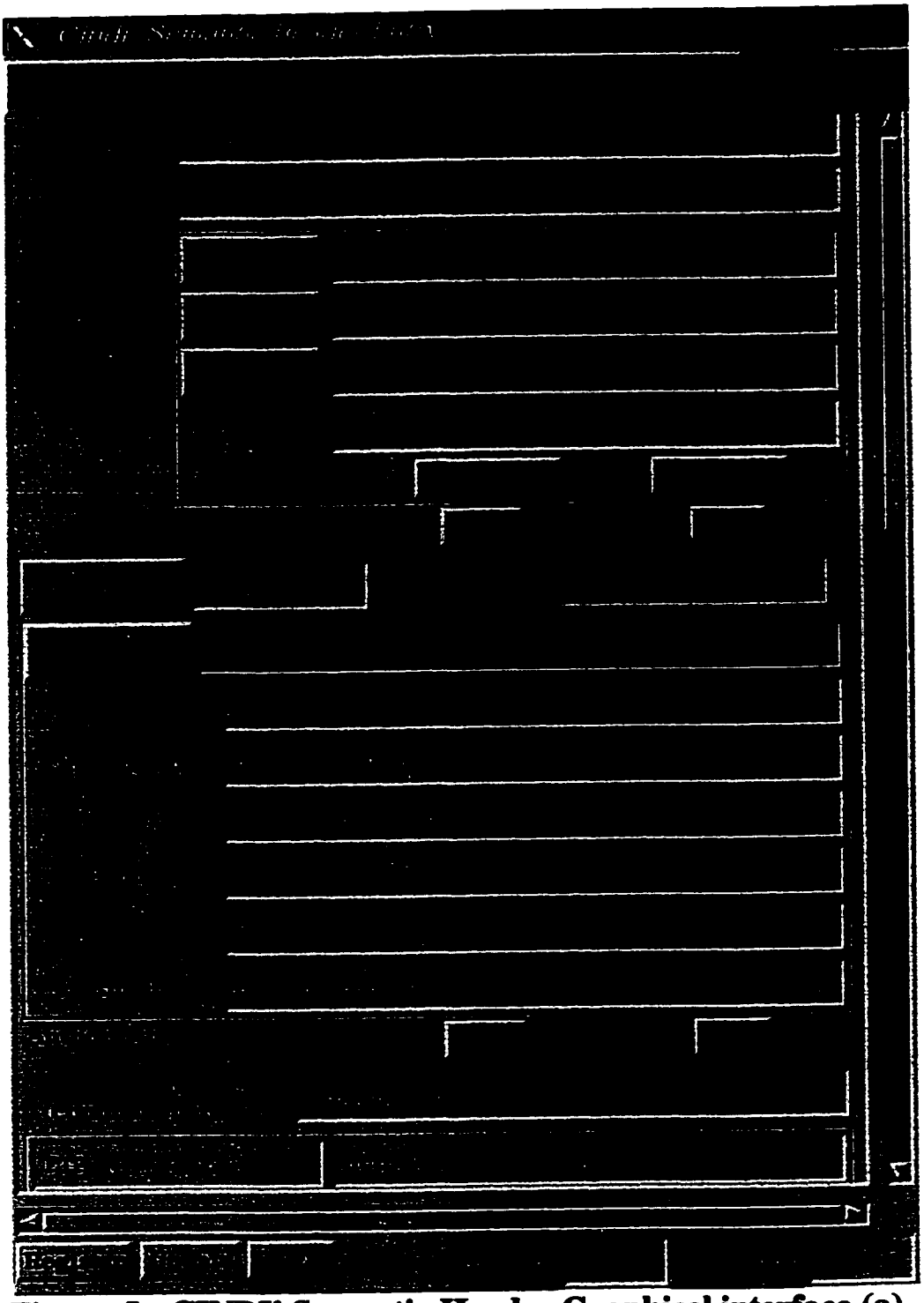

Figure 5 : CINDI' Semantic Header Graphical interface (a)

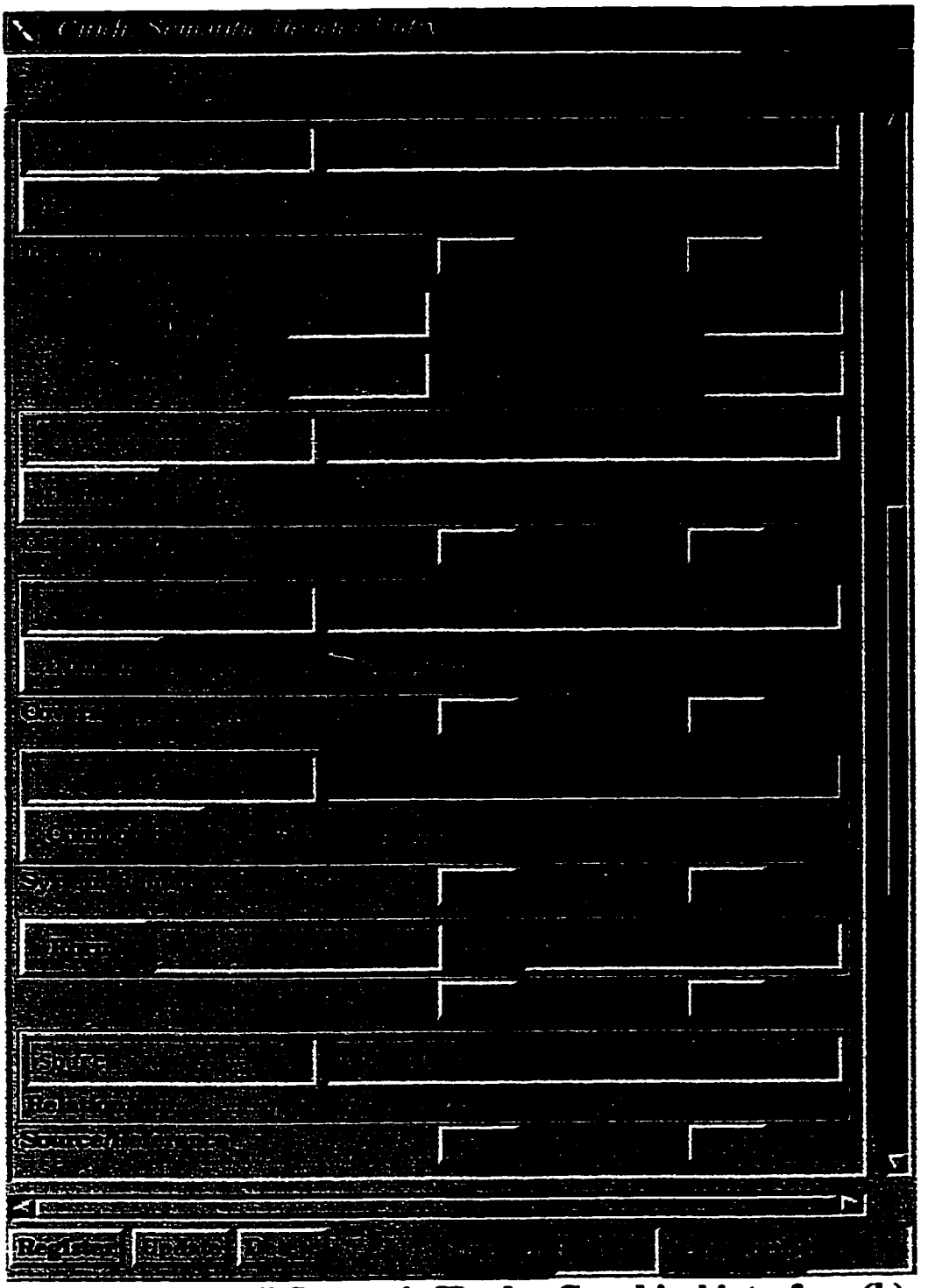

Figure 6: CINDI' Semantic Header Graphical interface (b)

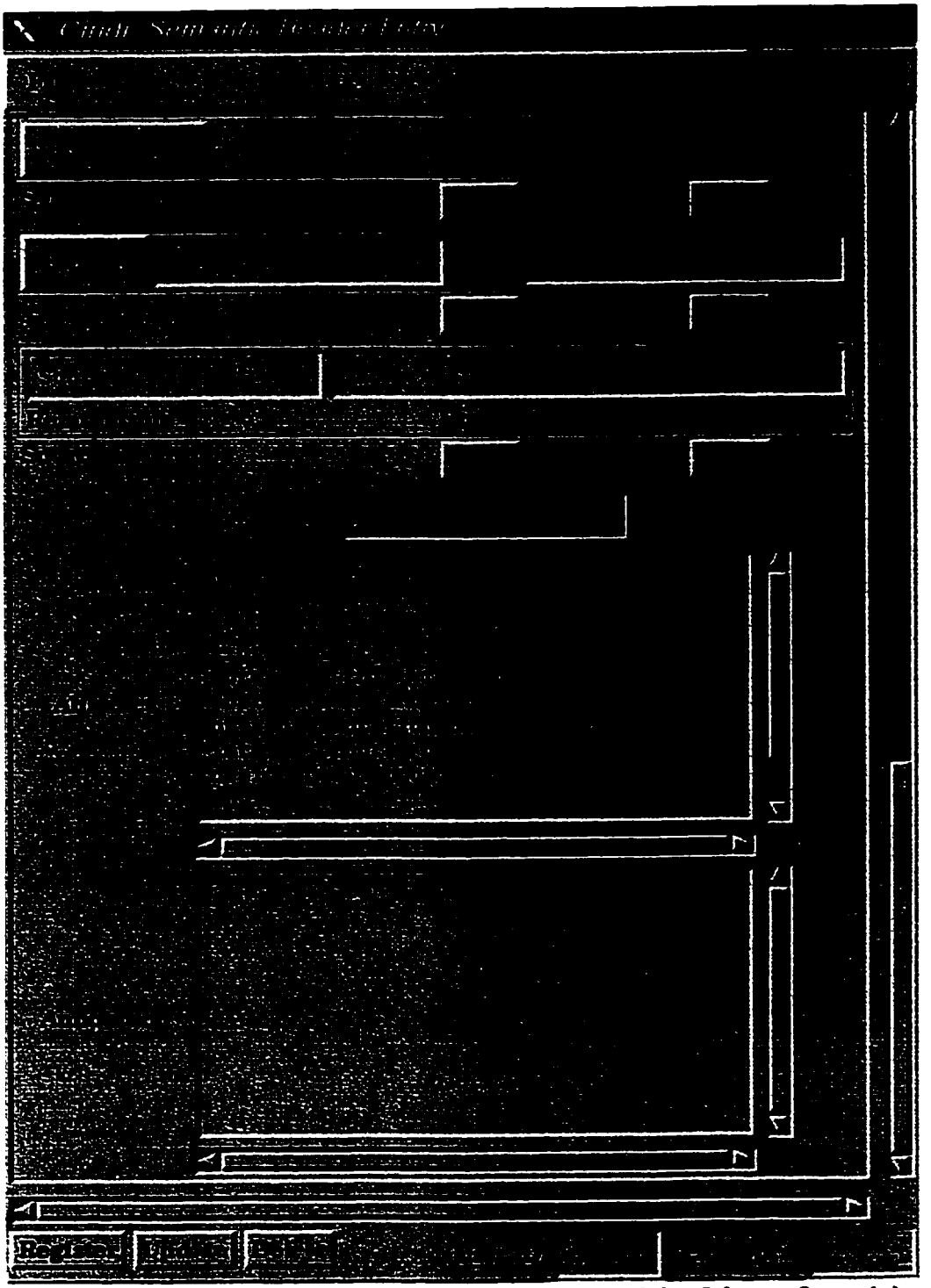

Figure 7 : CINDI' Semantic Header Graphical interface (c)

Semantic Header Registration

The system will perform a number of steps in order to register a Semantic Header. At the beginning, the parser will verify the syntax of the input file and make sure that the mandatory attributes of the Semantic Header have been entered. These attributes include those needed to assign a Semantic Header Name which are: title, first general subject, name of first author, date of creation, and finally version; as well as first identifier and one keyword. The non-noise words of the Semantic Header will be stored in temporary variables and data structures for later use. The next step would be for the database module to verify the status of the user ID and the password. Subsequently, it will be assured that such a Semantic Header does not exist in the database. Finally, the words and the Semantic Header are indexed into the database.

Semantic Header Deletion

A Semantic Header can be deleted by the user who registered the Semantic Header only if no annotation after the Semantic Header was registered. The procedure of deleting a Semantic Header is similar to that of registering it. The Semantic Header and the SHN maintained in the word object with the value corresponding to each non-noise word will be deleted from the database. If a word object happens to contain no SHNs, it will be deleted from the database as well.

Semantic Header Update

A Semantic Header can be updated by the user who registered the Semantic Header and requires the user ID and the password during the initial registration. However, a number of the attributes of a Semantic Header cannot be modified. These attributes are those comprising the Semantic Header Name as well as the annotation field added after the SH was registered. It updates those fields that are modified by the user. For each modified field, the deletion and addition of words are also managed by this method.

Chapter 6

6. Conclusion and Future Work

6.1 Conclusion

Content indexing provides a powerful means of helping users locate relevant information among large repositories. To be most effective, content indexing must exploit the semantics of the different types of data and different environments in which it is used. In this major report, we presented a model that will automatically extract and generate an index or meta-information. We have integrated this model which is called RTF extractor with ASHG.

ASHG exploits the file naming conventions and the data within a document to determine the document file type. ASHG exploits the semantics of the document's types in extracting the meta-information. It also assigns weights for terms depending on their location in the document. Both term weight and occurrence frequency was 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. CINDI has a 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.

The improvements made by the introduction of ASHG are that it extracts more information than other similar systems and it also tries to select the subject of the document by looking into our own thesaurus and the keyword-subject database.

Lastly, we applied ASHG to a collection of test RTF documents and compared the results to the actual assignments made by INSPEC. The results showed a 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.

6.2 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. For example, the term *wire* should not be extracted unless it is followed by another term such as *wire grid.* 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, and Mechanical Engineering. Extend the type of documents that ASHG can extract meta-information from.

Appendix

Papers Used in Testing ASHG for RTF_extractor

The following is the list of papers used in testing ASHG for RTF_extractor

Doc1 Grogono P., *Designing for Change*, Department of Computer Science, Concordia University, Montreal, Canada.

Doc2 Grogono P., *Designing a class library*, Department of Computer Science, Concordia University, Montreal, Canada.

Doc3 Grogono P., *A Code Generator for Dee*, Department of Computer Science, Concordia University, Montreal, Canada.

Doc4 Butler G., Grogono P., Shinghal R., and Tjandra I., *Retrieving Information from Data Flow Diagrams*, Department of Computer Science, Concordia University, Montreal, Canada.

Doc5 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.

Doc6 Grogono P. and Gargul M., *A Computational Model for Object Oriented Programming*, Department of Computer Science, Concordia University, Montreal, Canada.

Doc7 Grogono P. and Gargul M., *A Graph Model for Object Oriented Programming*, Department of Computer Science, Concordia University, Montreal, Canada.

Doc8 Grogono P. and Gargul M., *Graph Semantics for Object Oriented Programming*, Department of Computer Science, Concordia University, Montreal, Canada.

Doc9 Khorasani K., *Adaptive Control of Nonlinear Systems Using Output Feedback*, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada.

Doc10 Nascimento M. A. and Dunham M. H., *A Proposal for Indexing Bitemporal Databases Via Cooperative B+ trees*, Southern Methodist University, Dallas, USA.

Doc11 Grogono P., *Issues in the Design of an Object Oriented Programming Language*, Department of Computer Science, Concordia University, Montreal, Canada.

Doc12 Grogono P., *A Model for Computing with Objects*, Department of Computer Science, Concordia University, Montreal, Canada.

Doc13 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.

Doc14 Paknys R., *On the Accuracy of the UTD for the Scattering by a Cylinder*, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada.

Doc15 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.

Doc16 Grogono P. and Cheung B., *A Semantic Browser for Object Oriented Program Development*, Department of Computer Science, Concordia University, Montreal, Canada.

Doc17 Ounis I., Pasca M., *An Extended Inverted File Approach for Information Retrieval*, Grenoble, France.

Doc18 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.

Doc19 Nascimento M. A. and Dunham M. H., *A Proposal for Indexing Bitemporal Databases Via Cooperative B+ trees*, Southern Methodist University, Dallas, USA.

D20 Ludwig A., Becker P. and Guntzer U., *Interfacing Online Bibliographic Databases with Z39.50*, University of Tubingen, Germany.

Doc21 Park C. and Park S., *Alternative Correctness Criteria for Multiversion Concurrency Control and a Locking Protocol via Freezing*, Sogang University, Seoul, Korea.

Doc22 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.

Doc23 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.

Doc24 Ehikioya S. A., *A Formal Specification Strategy for Electronic Commerce*, Department of Computer Science University of Manitoba, Winnipeg, Manitoba, Canada.

Doc25 Revesz P. Z. and Li Y., *MLPQ: A Linear Constraint Database System with Aggregate Operators*, Dept. of Computer Science and Engineering, Lincoln, USA.

References

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

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

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

[4] 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

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

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

[7] Haddad S., ASHG: *Automatic Semantic Header Generator.* Master's thesis, Department of Computer Science, Concordia University, Montreal, Canada, 1998.

[8] 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.

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

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

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

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

[13] *Rich Text Format (RTF) Specification; RTF Version 1.3 documents.* http://night.primate.wisc.edu:80/software/RTF/

[14] Shayan N., CINDI: *Concordia INdexing and DIscovery system.* Master's thesis, Department of Computer Science, Concordia University, Montreal, Canada, 1997.

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

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

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

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

[19] 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.

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

[21] Youquan Zhou, CINDI: *The virtual Library Graphical User Interface.* Master's thesis, Department of Computer Science, Concordia University, Montreal, Canada, 1997.

[22] *UNIX Programming Perl*, 2nd Edition By Larry Wall, Tom Christiansen & Randal L. Schwartz 2nd Edition September 1996.

[23] *UNIX Learning Perl* Second Edition 1997 By Randal L. Schwartz and Tom Christianse.