

A GLEANING SUBSYSTEM for CINDI

TONG ZHANG

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE & SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTREAL, QUEBEC, CANADA

AUGUST 2004
© TONG ZHANG



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-94759-9

Our file Notre référence

ISBN: 0-612-94759-9

The author has granted a non-exclusive license allowing the Library and Archives 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 et Archives 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.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

A GLEANING SUBSYSTEM for CINDI

By
Tong Zhang

Internet search engines typically use Internet crawlers, or robots, for the purpose of constructing and maintaining a searchable index of resources on the Web. Most crawlers such as Google, Alta Vista, Excite, HotBot, and Lycos are designed to build global (non-topic-specific) indices of the resources on the Internet. Although the development of storage technology has made it possible to store large amounts of on-line data, the explosion of information available on the Web is overwhelming. Such search engines are exacerbating the problem by indexing more and more irrelevant documents. Therefore, topic specific robots will become popular in the next generation. They gather information on the Internet in specific domains by means of information filtering technology. The CINDI Robot System is such an application in academic domain.

This research is concerned with a structure-based gleaning subsystem for CINDI. The system separates theses, technical reports, academic papers, and FAQs as resources while e-mails, letters, resumes, graphics, and discussion groups are considered as chaff. This system makes decisions based on weight, which is carefully assigned to each resource by matching its structure with predefined Document Type Definitions (DTDs). The DTDs for the typical structure for the specific document types are built based on some predefined profiles. The test results from extensive experiment show that DFS has good performance.

The system also features conversion subsystem in Windows environment to unify document formats for CINDI. This subsystem converts non-pdf documents retrieved by the CINDI Robot into PDF. In this subsystem, a daemon was implemented to securely monitor the CINDI Robot database, automatically transfer new document between a Linux platform and a Windows platform, and maintain the converting information in the CINDI Robot database. The CINDI Robot database was designed and developed to store document information with other CINDI team members.

Acknowledgements

I would like to express my great appreciation to my supervisor, Professor Bipin C. Desai, for his thoughtful ideas, knowledgeable suggestions, and financial support. His extensive knowledge and lively work team will never be forgotten.

I would also like to thank the other members in the team, Hongbing Zhang, Cong Zhou, and Furong Xue. Helpful discussions with them have undoubtedly benefited my thesis. Their personalities make our cooperation effective and pleasant.

I would like to thank the staff on the helpdesk for their help in problem solving with the techniques, and equipment administrators for their excellent service. I am also grateful to the secretary of the department of Computer Science & Software Engineering, Halina Monkiewicz, for her help and enthusiasm during my whole study.

My special thanks go to Mary O'Malley in Student Learning Service in Counseling and Development for her impressive help in editing this paper.

I would also like to thank my parents although they are far from Canada. This thesis is the result of their unselfish encouragements and supports beginning from my childhood.

Finally, my great thanks go to my husband, Lianzhong Li, for his patience, works in household and continuous support and understanding. My gratitude also goes to my lovely daughter, Jiameng Li, for her understanding when I had no time to accompany her and to build a house with her.

Contents

List of Figures	vii
List of Tables.....	xiii
Acronyms	ix
Chapter 1	1
Introduction	1
1.1 Problem Statement	1
1.2 Proposed Solution	2
1.3 Organization of this Thesis	3
Chapter 2	5
Background	5
2.1 Typical Robot.....	5
2.2 Typical Documents on the Web.....	9
2.3 Filtering Technologies.....	10
2.4 Daemon	12
Chapter 3	13
File Conversion Subsystem and Document Filtering Subsystem	13
3.1 CINDI.....	13
3.2 The CINDI Robot Database	17
3.3 File Conversion System (FCS).....	19
3.3.1 PDF Converters	19
3.3.1.1 CZ-Doc2Pdf	19
3.3.1.2 PDFcamp	20
3.3.1.3 PDFlib	21
3.3.1.4 Pdf995	21
3.3.2 Evaluation of Converters.....	23
3.3.3 Configuration of FCS	25
3.3.4 Design of FCS	30
3.3.5 Implementation of FCSd	32
3.4 Document Filtering Subsystem (DFS)	38
3.4.1 The Approach.....	38
3.4.2 Document Type Definition (DTD).....	39
3.4.2.1 DTD for Theses	40
3.4.2.2 DTD for Technical Reports.....	42
3.4.2.3 DTD for FAQs	43
3.4.2.4 DTD for Academic Papers	44
3.4.2.5 DTD for E-mails.....	46
3.4.2.6 DTD for Resumes.....	46
3.4.2.7 DTD for Letters	47
3.4.2.8 DTD for Discussion Groups.....	48
3.4.3 Implementation of DFS	48

3.4.3.1 Thesis/Report Structure Extraction	49
3.4.3.2 Academic Paper Structure Extraction	56
3.4.3.3 FAQ Structure Extraction	60
3.4.3.4 Overall Algorithm for DFS	62
3.4.3.5 Data Structures for DFS	66
Chapter 4	71
Experiments and Evaluation.....	71
4.1 Experiment One on FCS	71
4.2 Experiment Two on DFS.....	74
4.2.1 Manual Test.....	75
4.2.2 Automatic Test	79
Chapter 5	85
Conclusion and Future work	85
5.1 Conclusion.....	85
5.2 Contribution of this thesis	86
5.3 Future Work	87
References	89
Appendix A Typical Document Types.....	93
Appendix B Source Formats Accepted by Omniformat and Pdf995	98
Appendix C Original MS Word Document Used for PDF Conversion Test	103
Appendix D Converted PDF Document Using PDFcamp	105
Appendix E Converted PDF Document Using Omniformat and Pdf995	107
Appendix F Key Generation for the Communication between Windows and Linux	109

List of Figures

Figure 2.1 Major Components of a Typical Robot	8
Figure 3.1 Architecture of CINDI	14
Figure 3.2 Amplification of the CINDI Robot database in CINDI Architecture	15
Figure 3.3 Communications between the Windows platform and the Linux platform.....	26
Figure 3.4 User Data Source in Control Panel	28
Figure 3.5 Configuration of MyODBC	29
Figure 3.6 Test Message of MyODBC.....	29
Figure 3.7 Architecture of File Converting System	31
Figure 3.8 Components of FCSd.....	373
Figure 3.9 Data Structure of Document List	37
Figure 3.10 Structure of a Block of Buffers.....	676
Figure 3.11 Sample of a Block of Buffers	67
Figure 3.12 Data Structure for the Body Structure of theses, technical reports, and academic papers	68
Figure 3.13 A Sample of the Body Structure for a Thesis	69
Figure 4.1 Conversion Flag in Database before PDF Conversion	71
Figure 4.2 The Starting Interface of FCSd.....	72
Figure 4.3 Document Transfer from the Linux Platform to the Windows Platform.....	72
Figure 4.4 The Interface of Document Conversion.....	73
Figure 4.5 Conversion Flag after PDF Conversion.....	73
Figure 4.6 Conversion Information Recorded in the CINDI Robot Database	74

List of Tables

Table 2.1 Development of Search Engines	6
Table 3.1 Comparison of Converters	23
Table 3.2 Samples of Document List	34
Table 3.3 Sample of Accepted Documents in the DOWNLOAD_STATUS Table	64
Table 3.4 Sample of the Rejected Documents in the DOWNLOAD_STATUS Table.....	65
Table 4.1 Test Document Contents for Manual Test of DFS	75
Table 4.2 The Results of Manual Test on DFS	77
Table 4.3 Results of Automatic Test on DFS	80
Table 4.4 The Filtering Accuracy of DFS from the Automatic Test	81
Table 4.5 The Filtering Accuracy after Tuning DFS	82
Table 4.6 Statistics on the precision of Relevant and Irrelevant Documents.....	83

Acronyms

ASHG	Automatic Semantic Header Generator
BMP	Bitmap format
CJK	Chinese, Japanese, and Korean
CINDI	Concordia Indexing and Discovery system
DFS	Document Filtering Subsystem
DOC	Microsoft Word documents
DSN	Data Source Name
DTD	Data Type Definition
FAQ	Frequently Asked Question
FCS	File Converting Subsystem
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol (World Wide Web protocol)
JPG	Joint Photographic Experts Group
MyODBC	MySQL ODBC driver
NLPQC	A Natural Language Processor for Querying CINDI
ODBC	Open Database Connectivity
PDF	Portable Document Format
PPT	MS PowerPoint documents
PS	PostScript documents
RTF	Rich Text Format documents
SHDB	Semantic Header Database
SQL	Structured Query Language
TEX	LaTeX format
TXT	Text format
VQAS	Virtual Query and Answering Subsystem for CINDI
WWW	World Wide Web
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Problem Statement

As the rapid growth of the World Wide Web (WWW) is providing a vast amount of information, Web search engines [1, 2] have become popular to provide information search accessible to a large number of naïve users. However, the difficulty for users is to choose a good search engine to help them easily and quickly find relevant information. Even Google, the most popular search engine among the top 10 search engines (HotBot, AltaVista, Northern Light, Excite, Infoseek, Lycos, Snap, Microsoft, Google, and Euroseek) [3], returns a large number of irrelevant pages. The problem is that Google tries to build indexes of all documents on all topics on the Web retrieved by web robots from Web servers. It returns the search results based on a few key words provided by the users. The explosive growth of the Web has resulted in billions of multimedia pages of information, including complete libraries of technical information, directories, personal web sites, entertainment, and advertisements which continue to be available via the Internet. Many of these pages that are not relevant to the user's search focus could be in the result of a search.

However, to produce more relevant search results, it is necessary to use a topic specific robot agent that automatically builds and maintains indexes of targeted document types on the Web. To focus on the significant documents, the system should be able to decide if information is relevant among a vast, unstructured, unorganized, and ambiguous mass of web pages. To solve this problem, we developed a Document Filtering Subsystem (DFS)

for CINDI (Concordia INDEXing and DIScovery system) [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14].

1.2 Proposed Solution

CINDI is a proposed digital library currently under development to provide fast and more accurate search results for specific academic disciplines. The bibliographic database in CINDI contains information either submitted directly by the authors or downloaded from the WWW sites. To obtain accurate information from the web sites, CINDI Robot gathers information from trusted web sites on specific topics. Then, DFS is employed to filter out irrelevant information from the retrieved documents. As an academic digital library, CINDI pays more attention to gathering academic documents such as theses, technical reports, academic papers, and Frequently Asked Questions (FAQs) on selected topics; CINDI's current focus is in the area of computer science. To obtain these relevant documents, DFS makes a filtering decision according to the document's structure by matching the candidate document with the predefined Data Type Definitions (DTDs). The DTDs for the typical structures for the specific types of documents were built based on the predefined profiles. DFS assumes the given document is a certain type of accepted document and then carefully assigns weight to the document based on the presence of key elements in it. The key elements are defined in the DTD for each relevant document type. The higher the weight, the closer the document is to the predefined document type.

Documents downloaded by CINDI Robot from web sites exist in many formats such as DOC, WPD, PPT, TXT, PS, HTML, RTF, XML, JPG, and LaTeX. Since DFS is based on text files, the downloaded documents must be converted into plain text files; such

conversion to text results in the loss of most formatting information such as font style, font size, and page layout. However, it is necessary for CINDI system to preserve the original format. The Portable Document Format (PDF) [15] has been chosen as the single document format supported by CINDI for two reasons. First, PDF is open standard and open source code software such as Xpdf [16] is available to convert PDF documents into plain text files. Second, PDF is portable across platforms. To convert documents that are platform specific such as those in DOC format, we developed a File Converting Subsystem (FCS) on a MS Windows platform. Cooperating with FCS, DFS is proposed to filter documents in many formats, as will be described in Chapter 3. To test the function of FCS, 194 non-pdf documents were converted by FCS. To measure the effectiveness of DFS, DFS was both manually and automatically tested. The difference between manual test and automatic test lies in the source data. In the manual test, the data source are man-made incomplete documents. The data source for the automatic test is from the Web, a real world. In the automatic test, 1003 documents retrieved by CINDI Robot were applied to. Comparison of the automatic test results and the manually checked results is discussed in Chapter 4.

1.3 Organization of this Thesis

This thesis is organized as follows. In Chapter 2, the concept of Web robots, typical documents on the Web, and related filtering techniques are presented. Chapter 3 illustrates the detail design and implementation of the Gleaning Subsystem made up of a document converter (FCS) and a filter (DFS). The experiments using FCS and DFS, and the evaluation are presented in Chapter 4. In Chapter 6, we give our conclusion,

contribution of this thesis, and suggestions for future research related to Web-based topic
specific robot systems

Chapter 2

Background

2.1 Typical Robot

The Web search engine, as an information search tool, is primarily concerned with two distinct processes: indexing and ranking [17]. Indexing of the available documents on the Web makes information retrieval more efficient. Ranking returns a list of the most relevant documents in response to a given query.

Document acquisition of a search engine can follow either a push or pull model [4]. In the push model, contributors submit documents to a search engine for indexing. In the pull model, search engines acquire documents through a robot. A robot [4, 18, 19, 20, 21, 22, 23, 24, 25] is a program that traverses the Web's hypertext structure by retrieving a document, and recursively retrieving all documents that it references [18]. The program is sometimes called spider, Web wanderers, web crawler, or web worms. Beginning with the first Web worm in 1990, a large number of crawlers have been developed, as described in Table 2.1 [26].

The common genesis of the robots listed in Table 2.1 is that all of them were initially from university research. For example, Yahoo was primarily developed by David Filo and Jerry Yang, two Ph.D. candidates in Electrical Engineering at Stanford University [26]. They started their guide in a campus trailer in February 1994 as a way to keep track of their personal interests on the Internet.

Table 2.1 Development of Search Engines

	Foundational technology	Net-Oriented Access	Spider	Test-Content Indexing	Browsers	Directories	Meta- Search	Person alized Organi zation	Qualifie d Result
Pre 1990	ARPANet; AOL								
1990	HTTP; URL; WWW; HTML	Archie							
1991		Gopher							
1992		Veronica							
1993		Jughead; ALIWEB; Jumpstation	WWW Wanderer; WWW Worm; RSBE Spider	Architext; exite	Mosaic				
1994				Webcrawler; Lycos; Infoseek; Altavista	Netscape	Galaxy; Yahoo!; N.Light			
1995					IE		Savvy Search; Meta- crawler		
1996				HotBot; Inktomi; Infospace		LookSmart		Ask Jeeves	
1997				GoTo; Overture					
1998						Open Directory Project			Google; Direct Hit
1999				All the Web; FAST					

Similarly, Google was developed by Sergey Brin and Larry Page, two Ph.D students at Stanford University in 1998 [27, 28]. The earliest crawlers mentioned in Table 2.1 that still exist in some form as part of current search engines are listed as follows:

Excite	(1993)
Galaxy	(1994) directory
Yahoo	(1994) directory
WebCrawler	(1994) first full-text search
Lycos	(1994)
Infoseek	(1994)
Altavista	(1995) first natural language queries, boolean operators, link search
Metacrawler	(1995)
Inktomi/Hotbot	(1996)
Google	(1998)

The typical robot is implemented as a software system that retrieves information from remote sites using standard HTTP protocols. The structure of the Web is similar to a directed graph, so it can be traversed using graph-traversal algorithms. There are currently three approaches for traversal [29]:

- (i) Providing the robot a “seed URL” to initiate exploration. The robot retrieves the seed documents and extracts URLs pointing to other documents. Each of these URLs is used recursively in a breadth-first or depth-first fashion.
- (ii) Starting with a set of URLs determined on the basis of a Web site’s status and searching recursively.
- (iii) Partitioning the Web space based on the Internet names or country codes and assigning one or more robots to explore the space exhaustively. This method is more widely used than the first two.

A typical robot has the components shown in Figure 2.1.

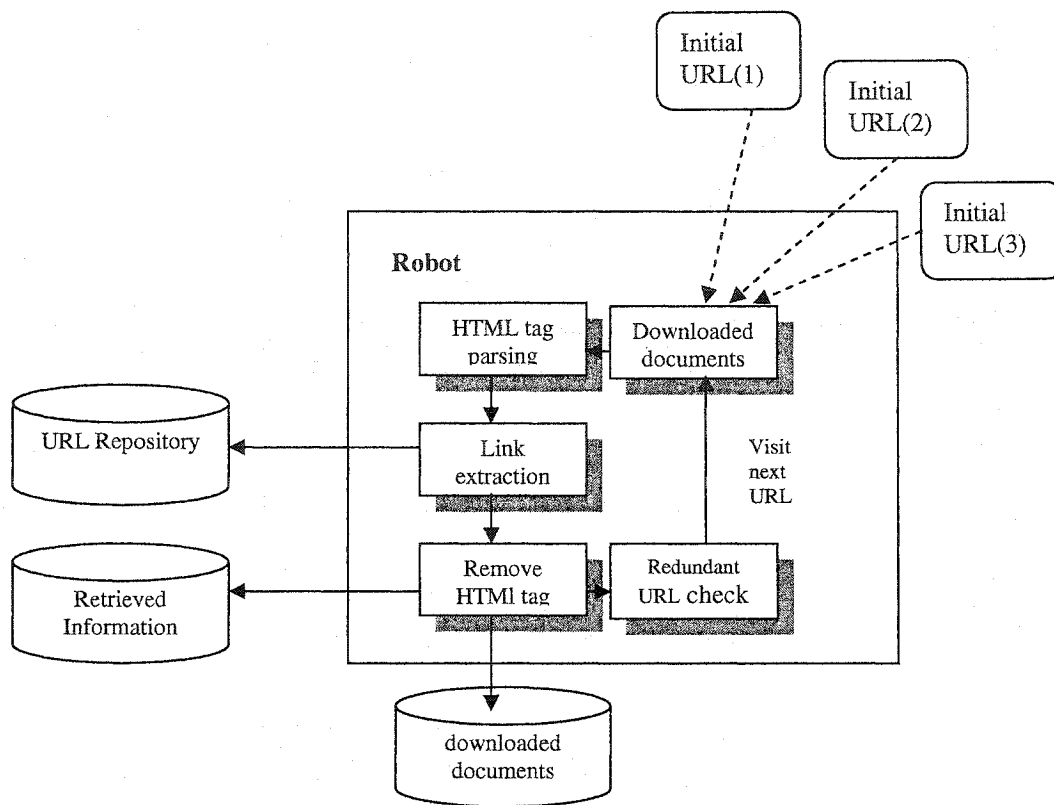


Figure 2.1 Major Components of a Typical Robot

From Figure 2.1, a robot downloads an initial set of URLs and parses the associated documents to extract additional URLs pointing to other documents. Before the robot downloads documents to the repository, it converts all relative URLs hidden in tags, such as `<IMG...>`, `<A HREF...>`, `<APPLET...>`, and `<AREA...>`, to absolute URLs and stores them in a URL repository. After that, the robot checks the extracted URLs to determine if they were already visited and then try to access the ones that were not previously retrieved.

Due to the fact that Web robots can place high demands on a Web server, many sites use the Robot Exclusion Protocol to limit the use of automated retrieval tools. In practice, Web sites place a robots.txt file in the site's root directory specifying which areas of the

site are off limits to robots. Web robots in turn voluntarily abide by these restrictions [30].

2.2 Typical Documents on the Web

Information on the Internet from commercial companies, government organizations, research institutes, as well as personal websites are growing rapidly. Early in 1999, Lawrence and Giles [31] estimated that the number of publicly accessible pages on the Web was about 800 million (with a total of 6 terabytes of data) on about 3 million servers. Today, the Web contains about 100TB of information [30]. Given the enormous volume of Web pages in existence, it comes as no surprise that the data types for this information is varied. The reason is not only almost all genre of documents shown in Appendix A (237 types) can be found on the Web; additionally, many other genre of documents such as FAQ and group discussion also exist on the Internet.

The typical documents on the Web include seminar announcements, job listings, Ads, e-mails, forms, letters, pictures, group discussions, News, FAQs, as well as scientific papers. Since CINDI concentrates on academic and learned resources, most of the above documents and other non-desirable resources are excluded. In addition, since group discussions typically contain repeated queries and unreliable responses, they are not a good resource for CINDI.

2.3 Filtering Technologies

Filtering as applied to information specifically is a process of comparing an incoming document to the profile of a user's interests and ranking it according to that profile [2, 32]. A filter [23, 27, 32, 33, 34, 35, 36, and 37] can be implemented in three ways.

One way is by correlating users' ratings of documents. In this approach, a document is recommended to a user because it is highly rated by other users with whom they tend to agree. This can also work for negative ratings; an article may not be recommended because some other "colleagues" did not like it. To make the system more precise and useful over time, it is important to enrich the initial profile with user feedback. Examples of such collaborative systems are GroupLens [33], Ringo [34], and WAIR [35].

Another way of filtering is the content-based collaborative filtering technique [23, 32, 36, 37]. In this technique, document contents are utilized in filtering. Chun-sheng et al.[23] present a content-based filtering technique, called Information Filtering (IF) agent, in their information gathering system. The IF agent first determine information categories according to an expression. The expression consists of some sequential words with logic operators from the user's query. The information categories are determined by the user based on the frequency of his or her expression appearing in each category. Finally, IF filters the rough documents based on the frequency of the expression appearing in the documents in the selected categories.

The third way for document filtering is a structure-based filtering technique. Autonomous Citation Indexing (ACI) system is an example of this technique [38]. It can automatically create a citation index from literature in electronic format to allow the user to navigate

the literature backward in time (through the list of cited articles) or forward in time (to find more recent, related articles). CiteSeer is an example of such a system; it collects research documents from the Web and filters them for reference or bibliography sections. Then, it parses each citation in the reference list using heuristics(syntax, position, and composition) to extract fields such as title, author, year of publication, page numbers, and the citation identifier. As a digital library, CiteSeer also uses other technologies for its search purpose which is not discussed here because it is not related to the filtering technology.

Recall that the purpose of DFS in CINDI Robot Subsystem is to filter out irrelevant documents from a set of downloaded files. Unlike the content-based filtering in the IF agent which needs to know the content of documents and make multiple decisions for categorizing them, the decision process of DFS is binary: accept or reject. Therefore, DFS was developed using a structure-based filtering technique.

Unlike CiteSeer whose purpose is to collect scientific articles with citation, the goal of CINDI is to collect scientific documents including those without citation such as FAQs. Therefore, DFS filters documents by extracting the whole document structure according to predefined DTDs. The DTDs define the typical structure for each of the desirable document types. DFS makes filtering decisions based on weight, which is carefully assigned to each downloaded document by matching its structure with the DTDs. The detail design and implementation of DFS is given in Chapter 3.

2.4 Daemon

A daemon is a background process ready to perform an operation when required. Functioning as an extension of the operating system, the daemon is an unattended process that usually is initiated at system startup. Typical daemons are print spoolers, e-mail handlers, and a scheduler that starts up another process at a designated time. The term daemon comes from Greek mythology meaning "guardian spirit." Daemons spend most of their time sleeping until something comes along which requires their help. A UNIX system has a number of daemons. For instance, the cron daemon automatically runs shell commands at specified dates and times. In order to automatically monitor the CINDI database and securely transfer available documents between a Linux platform and a Windows platform, a daemon called FCSd runs on a Windows platform. FCSd examines the CINDI database every thirty minutes; it transfers a batch of the non-pdf documents from the Linux platform to the Windows platform for conversion and sends the converted PDF files from the Windows platform back to the Linux Platform.

Chapter 3

File Conversion Subsystem and Document Filtering Subsystem

3.1 CINDI

CINDI utilizes a robot to retrieve information from the Web for subsequent filtering. After converting the documents to a single format and filtering out irrelevant documents, the system populates the accepted documents into a repository for full indexing by Virtual Query Answering Subsystem (VQAS), a subsystem of CINDI, and for indexing carried out by another CINDI subsystem called Automatic Semantic Header Generator (ASHG). ASHG indexes the primary information into the bibliographic database. The architecture of CINDI is shown in Figure 3.1. The highlighted part, FCS and DFS, are the main contribution of this thesis. Figure 3.2 is the amplification of the CINDI Robot database in Figure 3.1.

As illustrated in Figure 3.1, CINDI accepts, via the CINDI Registration Subsystem, documents in HTML, TXT, LaTeX, RTF, and PDF formats, contributed directly by authors; it also uses a robot to find documents on the WWW. In addition, it uses papers submitted to ConfSys, a conference management subsystem of CINDI. The documents directly registered by authors or through ConfSys are directly stored in the Document Collection repository. To find documents on the Web, Seeds Finder extracts links (seeds) from a search result of the AltaVista (Yahoo) search engine for queries such as “computer science department”, “computer science research page”, or “computer science publications”. Then, the Seeds Finder stores the extracted URLs in the SEED_URL table as shown in Figure 3.2. These seeds are the starting URLs for the CINDI Robot. The

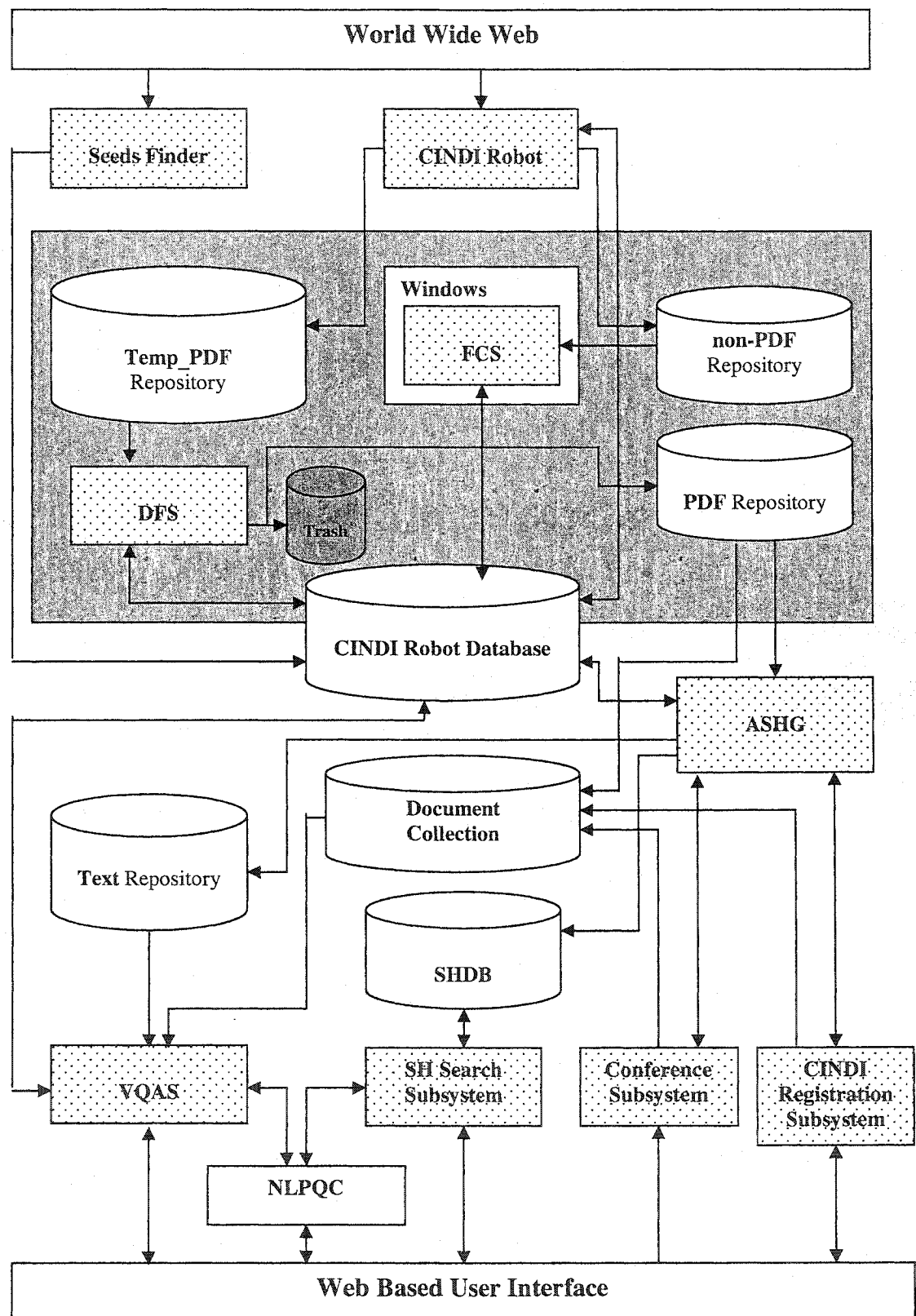


Figure 3.1 Architecture of CINDI

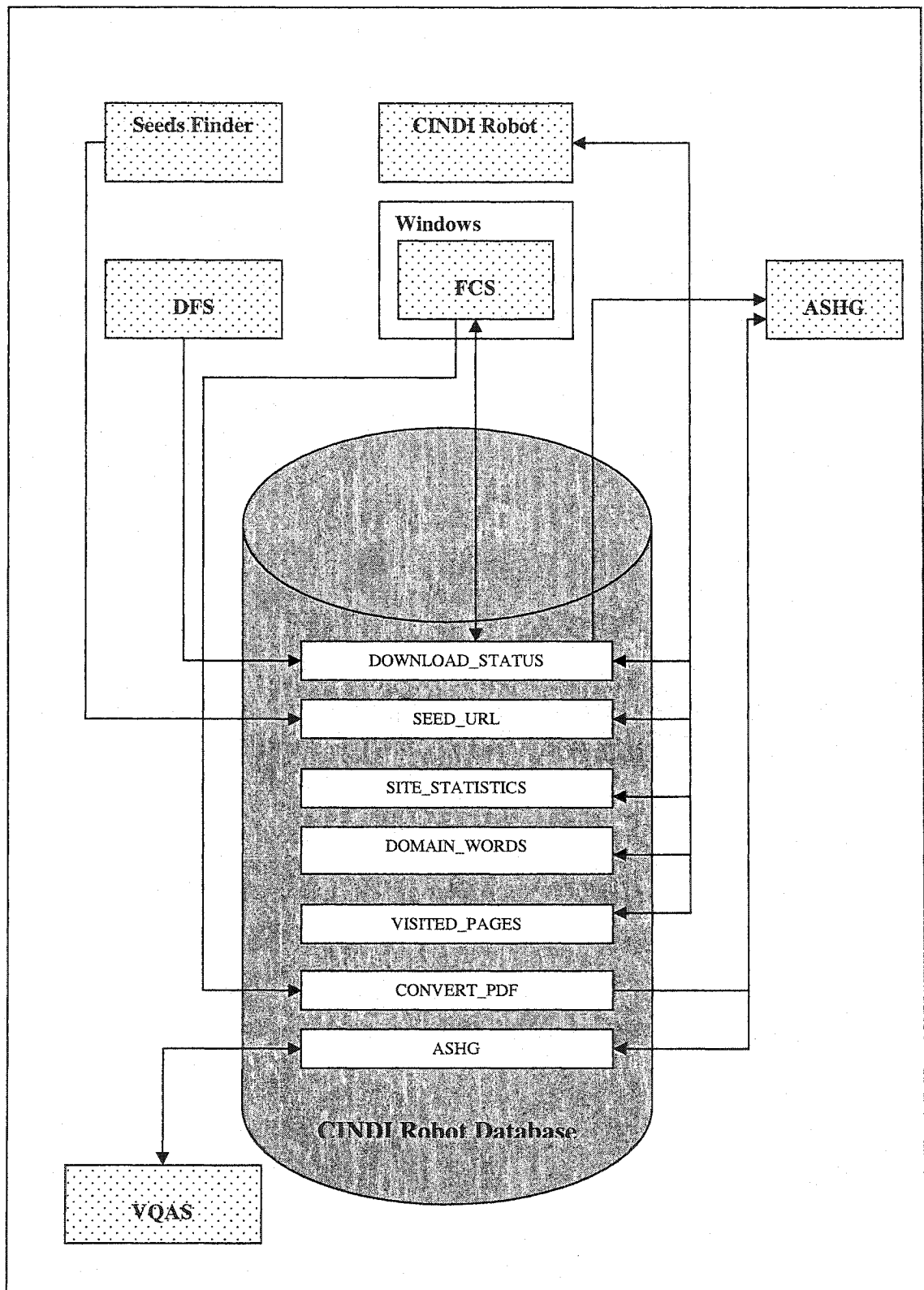


Figure 3.2 Amplification of the CINDI Robot database in CINDI Architecture

CINDI Robot retrieves seeds from the SEED_URL table, downloads the corresponding pages, and extracts and follows the links from these pages. It downloads the documents from the visited pages to CINDI temporary repository, storing the visited URLs in the VISITED_PAGES table, as well as the download status in the DOWNLOAD_STATUS table as shown in Figure 3.2. The documents in PDF format are stored in the temporary Temp_PDF Repository; the other documents are saved in the non-PDF repository. The non-PDF documents are converted to PDF format by FCS and then passed to DFS for filtering. DFS determines document quality by checking if it is accepted or rejected and saves quality information in the DOWNLOAD_STATUS table. The irrelevant documents such as emails, letters, news, pictures, assignments, application forms, slides, and others are rejected and are thrown into the CINDI trash. Relevant documents such as theses, technical reports, academic papers, and FAQs are accepted for CINDI and are stored in the permanent PDF Repository. In the next version of CINDI, the PDF Repository and Document Collection are to be merged. The CINDI Robot calculates the scores for each major URL directory or server based on the number of valid documents as determined by DFS and stores them in the SITE_STATISTICS table. This score determines the frequency of the CINDI Robot if it revisits a given site. Before the next crawling, the CINDI Robot also extracts key words from the relevant URLs with high scores and stores them in the DOMAIN_WORDS table. In the consecutive crawling, a URL not having high score but having high similarity to the key words in the DOMAIN_WORDS table will be frequently visited by the CINDI Robot.

As shown in Figure 3.1, ASHG uses the accepted documents from the PDF Repository to generate the corresponding semantic header [5, 14] which includes title, author, subject,

and abstract, and stores these into the SHDB database. In order to extract the semantic header from the PDF documents, they are converted into plain text file. ASHG also saves the converted text files in the Text Repository. These text files are used by the VQAS to create indexes for virtual query and answer. A natural language processing interface to users' queries is provided by NLPQC (A Natural Language Processor for Querying CINDI), another CINDI subsystem. VQAS has an interface to accept users' queries processed by NLPQC. VQAS returns the most relevant contents from the Text Repository through the VQAS indexes. VQAS also provides a link for users to the PDF version of the document in the Document Collection repository. Users can also query SHDB using author, title, subject, or keywords utilizing the CINDI Search Subsystem, another CINDI project. The CINDI Robot database stores information regarding the retrieved documents, their URLs, locations, the format of the documents, and the semantic headers for these documents. The CINDI Robot, FCS, DFS, ASHG, and VQAS share the information in this database.

3.2 The CINDI Robot Database

MySQL is an open source relational database management system that uses SQL for querying the data. MySQL, which is actively maintained, provides APIs for programming languages such as C, C++, Eiffel, Java, Perl, PHP, and Python, while having many features such as multi-users and multi-thread. Due to these attractive features, CINDI uses MySQL as its database server.

The CINDI Robot database contains seven major tables: SEED_URL, DOWNLOAD_STATUS, SITE_STATISTICS, DOMAIN_WORDS, VISITED_PAGES,

CONVERT_PDF, and ASHG. Information about the starting urls, file name, download date, document format, statistics on page quality, domain key words, visited pages, PDF conversion flag, date of conversion, and file location for each downloaded document are recorded in these tables. The first five tables are created by the CINDI Robot and the second table is shared by FCS, DFS, ASHG, and VQAS. The table CONVERT_PDF is created by FCS and utilized by FCS, DFS, ASHG, and VQAS; the table ASHG is created by ASHG and shared by ASHG and VQAS.

The Seeds Finder extracts links from AltaVista (Yahoo) search engine and stores the links in the SEED_URL table. The CINDI Robot uses the links in SEED_URL as starting points to download documents and record the file name, url, document type, and temporary file location in the DOWNLOAD_STATUS table for each downloaded document. At the same time, the CINDI Robot stores the visited URLs in the VISITED_PAGES table. FCS sets the conversion flag in DOWNLOAD_STATUS to "TRUE" and writes the file name of the document that is converted into PDF format, date of conversion, temporary location of the converted document, and the document ID referenced in DOWNLOAD_STATUS in the CONVERT_PDF table. DFS sets the final location (the location after filtering) and filtering flag for each downloaded document in the DOWNLOAD_STATUS table to indicate whether the document is accepted or not. The CINDI Robot computes various statistics based on the result of filtering and writes the results in the SITE_STATISTICS table. For the valuable pages with many accepted documents, the CINDI Robot extracts key words from these pages and store them in the DOMAIN_WORDS table. The Robot would avoid, for example, sites with a high percentage of rejected documents. ASHG creates a semantic header for each accepted

document and stores the file name of the semantic header, time of creation, updating time, the location of the semantic header, and the name and the location of the converted text file in the ASHG table.

3.3 File Conversion System (FCS)

In this section, we first introduce and compare four PDF converters, and explain our choice for use in FCS. Then, the configuration of FCS is introduced. Finally, the design of FCS for CINDI is presented. The design and implementation of the FCSd daemon that transfers files between a Windows platform and a Linux platform is given in section 3.3.5.

3.3.1 PDF Converters

PDF has become a popular portable e-document format [14] for which there are a number of converters available. For our purpose, an ideal PDF converter for CINDI should have the features of excellent converting quality, flexibility, the ability to convert different document formats, as well as command line executability. Among current PDF converters, typical examples that could be used for this project are CZ-Doc2PDF [39], PDFcamp[40], Pdflib[41], and Pdf995[42].

3.3.1.1 CZ-Doc2Pdf

CZ-Doc2Pdf is a batch PDF creator that converts DOC, HTML, TXT, and RTF to PDF. It can monitor the source file folder and convert MS word, RTF, TXT, or HTML documents to PDF files automatically. CZ-Doc2Pdf has the following key features:

- Creates PDF in batch mode for DOC, html, text, and RTF files

- Preserves original document layout including URL links, bookmarks, images and tables.
- Controls the settings for Distiller and/or PDFMaker.
- Supports command line arguments.
- Puts the files into subfolders for converting.
- Supports drag and drop of files.
- Creates run log file.

3.3.1.2 PDFcamp

PDFcamp (PDF writer) is a product of Vervypdf Company. It is PDF creation software that converts a printable Windows document in DOC, XLS, PPT, TXT, or HTML format to PDF format. The newest version of PDFcamp Pro (Ver 1.9) has the following features:

- Supports three modes for creating PDF files: user interface to manually select output filename, automatic conversion of many files to PDF at one time, and BatchToPDF software to create PDF files using command line input without users' intervention.
- Espouses Windows applications including Visual Basic, Visual C++, and Delphi, in which the PDF file name can be managed without user intervention
- Supports Text Extraction from printable documents (except for graphics and PDF files) and keeps the original page layout. The extracted text is independent of the software that created the original document and can be used to re-construct the document and/or be inserted into a searchable text database. Text Extraction is ideal for archiving form documents such as invoices, statements, and reports.

- Integrates with Microsoft Office 2000 and creates toolbars and icons in Microsoft Office 2000

3.3.1.3 PDFlib

PDFlib is a development tool for generating PDF documents on a server. It offers a simple-to-use API for creating PDF files within a user's server-side or client-side software. In addition, it does not make use of third-party software for generating PDF, nor does it require any other tools. The features of PDFlib are:

- Completely reworked font engine with full Unicode and CJK (Chinese, Japanese, and Korean) support.
- BMP image support and image formatting options.
- Smooth shadings, transparency, blends, and overprint control to graphics.
- Hypertext including named destinations, bookmark targets, and improved coordinate handling.
- 40-bit and 128-bit encryption for user and mastering password and permission settings.
- New exception handling for C and consistent numbering of error messages.

3.3.1.4 Pdf995

Pdf995 makes it easy and affordable to create professional-quality documents in PDF format. It offers the following features:

- Automatic insertions of the embedded links and hierarchical bookmarks.
- Supports for digital signatures and Triple DES encryption.
- Batch prints from Microsoft Office.

- Supports for large format architectural printing in XP Fast User Switching and multiple user sessions, Optimized PDF, Citrix/Terminal Server, and Windows 2003 Server.

Apart from the features above, Pdf995 is also capable of carrying out the following functions:

- Automatically generates Table of Contents.
- Appends and deletes PDF Pages.
- Supports Asian and Cyrillic fonts.
- Combines multiple PDF documents into a single one.
- Impositions Draft/Confidential stamps.
- Has standard PDF Encryption (restricted printing, modifying, copying text and images).
- Has options to attach PDFs to email or automatically displays PDFs after creation.
- Automatically generates page numbering and text summarization of PDF documents.
- Integrates easily with document management and Workflow systems.
- Simplifies programmer interface.
- Customizes sizes of PDF output.
- Configures Font embedding.
- Has an executable program to specify PDF document properties.
- Masters PDF opening mode.
- Creates PDF documents without annoying watermarks.
- Is free with full functions but has sponsor pages.

Cooperating with Omniformat, which is another free software of Pdf995 Company, Pdf995 can convert over 75 file formats including MS Office and Word Perfect formats to PDF as described in Appendix B. Omniformat can monitor the specific folder and automatically convert newly downloaded documents to PDF files at any time. Also, it may be used to dynamically convert XML data to any supported output format including PDF. In the process, a Microsoft Word file is utilized as a template to supply layout information. Moreover, Omniformat replaces fields in the Word template with values specified in the XML file and converts the document to the final format.

3.3.2 Evaluation of Converters

In FCS, the requirement for a converter is that it can automatically convert, with high fidelity, a batch of files with many formats without user's interaction. Each converter above has its own features; the comparison of the four converters is presented in Table 3.1.

Table 3.1 Comparison of Converters

	CZ- Doc2Pdf	PDFcamp	PDFlib	Pdf995+Omniformat
Accept format	Doc, html, txt, rtf	doc, xls, ppt, .html, .txt, .rtf, .tif, .jpg	rtf, ttc, ps, bmp	html, doc, xls, wpd, jpg, gif, tif, png, pcx, ppt, ps, txt, Photo CD, FAX and MPEG See Appendix B.
Automatically converting	Yes	Yes	Yes	Yes
Support command line	Yes	Yes	Yes	Yes
Font embedding	No	No	No	Yes
Batch converting	Yes	Yes	No	Yes
Support languages	C/C++	VB, VC++, Delphi	Cobol, COM, C, C++, Java, .NET, Perl, PHP, Python, RPG, Tcl	C/C++, VB, .NET
Open source	No	No	No	Yes
Free	No	No	No	Yes

It can be seen in Table 3.1 that PDFlib does not support batch converting; therefore, it cannot meet one of the requirements of FCS. Even though CZ-Doc2Pdf can convert batches of files automatically, it has the drawbacks of being able to convert only a limited number of document formats, and being neither open source nor free. PDFcamp converts more document formats than CZ-Doc2Pdf and supports automatic conversion of batches of files. However, the source code of PDFcamp is not available. Thus, it is not flexible to improve its functionality so as to integrate the converter into FCS. PDFcamp is not free and the trial version creates a red watermark consisting of the company's URL address on each page of the converted PDF document. Significantly, PDFcamp's conversion quality is poor. This is demonstrated by the conversion of the document given in Appendix C; the result of the conversion is given in Appendix D. To test the conversion quality of PDFcamp, the test document of Appendix C was made up of five components: a paragraph with table, figure, some special symbols, hyperlinks, as well as a JPG image. First of all, as shown in Appendix D, some special symbols such as Π , ω , ν , δ , ρ , \leftarrow , \downarrow , ∇ , \neq , \in , \therefore , and \therefore , could not be recognized in the conversion. Pdfcamp replaced those unrecognized symbols with question marks "?". Secondly, some converted symbols were totally wrong, e.g. δ was converted as d, and σ was converted as s, while the symbol σ in the given figure was lost. Finally, it could not identify hyperlinks in the PDF document since it only converted hyperlinks to a plain text.

In contrast to PDFcamp, pdf995 has excellent converting quality as presented in Appendix E. Based on the same test document of Appendix C, Pdf995 recognized and converted special symbols correctly. Furthermore, Pdf995 can identify hyperlinks in the converted PDF file. A user can connect to the link by clicking the hyperlink in the

converted PDF file. Also, there is a popup box to show the full URL address. Most significantly, Pdf995 is open source with extensive documentation. Therefore, it is convenient and flexible to improve its functionality to meet our needs. In addition, Pdf995 has extra robust functionalities such as supporting Asian and Cyrillic fonts, configurable font embedding, and standard PDF encryption, which are benefit for the future improvement of CINDI.

Consequently, comparing converters based on conversion quality, accepted document formats, flexibility, and availability, the cooperation of pdf995 and OmniFormat is an ideal choice for FCS due to its features of open source, low cost, convenient functionalities, excellent converting quality and speed, and the number of source document formats supported.

3.3.3 Configuration of FCS

The software OpenOffice can open and convert DOC documents into PDF on a Linux platform. However, the conversion quality and ability of OpenOffice is often lags the frequent updating of Microsoft Word. Therefore, FCS was built on a Windows 2000 Professional platform. Since CINDI is built on a Linux platform, non-PDF documents in such a format downloaded by the CINDI Robot must be transferred to a Windows platform for conversion. Then the converted files are sent back. Therefore, FCS uses a daemon called FCSd, which is responsible for the file transfer. The detail design of FCS and the implementation of FCSd is described in sections 3.3.4, 3.3.5, and 3.3.6.

To ensure that all documents are transferred securely across platforms, FCS is developed as a secure server and employs a communication software for file transmission.

Communication software, such as ftp, is not secure for transferring files. In addition, the user's password is transmitted without encryption, and the communication requires user interaction. In contrast to ftp, OpenSSH [43] is a free version of the SSH protocol suite of network connectivity. It encrypts all traffic including passwords to eliminate eavesdropping, connection hijacking, and other network-level attacks. Moreover, it provides secure tunnel as well as authentication methods.

Since OpenSSH provides cryptography and authentication for secure Internet communications, we complemented a secure environment in a Windows platform based on OpenSSH server. Public key and private key were generated as illustrated in Appendix F, and the public key was copied to the Linux platform so that the FCSd could automatically communicate with other subsystems of CINDI without the requirement of entering a password. In the environment illustrated in Figure 3.3, documents can be securely and automatically transferred between the Linux platform and the Windows platform.

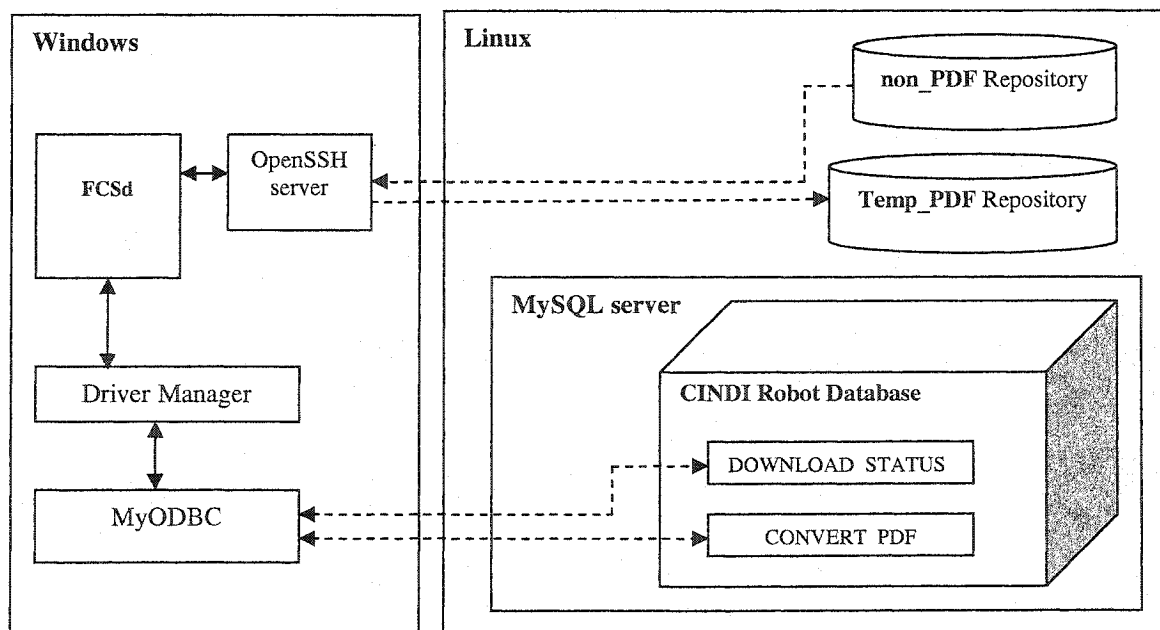


Figure 3.3 Communications between the Windows platform and the Linux platform

As shown in Figure 3.3, FCSd runs on the Windows platform, while the CINDI Robot database described in section 3.2 was created on a MySQL server running on the Linux platform. All non-PDF documents downloaded by the CINDI Robot are stored in the non_PDF Repository. The Temp_PDF Repository holds temporary PDF documents consisting of PDF files directly retrieved by the CINDI Robot and the converted PDF files converted by FCS. Both non_PDF and Temp_PDF repositories are located on the Linux platform; non-PDF documents from the non_PDF repository are safely transferred to the repository on the Windows platform by FCSd through OpenSSH server using pscp command and vice versa. Apart from file transaction, FCSd requires to record information in the CINDI Robot database for the converted documents. In Windows 2000 environment, MS ODBC is available for APIs to directly communicate with Oracle, SQL Server, Access, Excel, dBase, and FoxPro but cannot communicate with MySQL. Fortunately, MySQL supplies a convenient bridge, the MyODBC driver, between the Driver Manager and MySQL server. The application FCSd communicates with the Driver Manager and MyODBC driver directly using the standard ODBC calls. The FCSd only needs to know the Data Source Name (DSN). Therefore, the MyODBC driver was installed on the Windows 2000 platform and added into ODBC Data Source Administrator shown in Figure 3.4. Before using MyODBC to access the databases on MySQL server, MyODBC needs to be configured into ODBC Data Source Administrator denoted in Figure 3.5. The following steps complete the configuration shown in Figure 3.5:

1. In the Data Source Name (DSN) box, type the name of the data source you want to access. It can be any valid name.

2. In the Description box, type the description required for the DSN.
3. In the Host or Server Name (or IP address), type the MySQL server name. By default it is 'local host'.
4. In the Database Name, type the name of the MySQL database to be the default database.
5. In the User box, type database user name (database user ID).
6. In the Password box, type the user's password.
7. In the Port box, type the port number if it is not the default (3306).
8. In the SQL Command box, enter the optional SQL command for testing to be used after the connection.

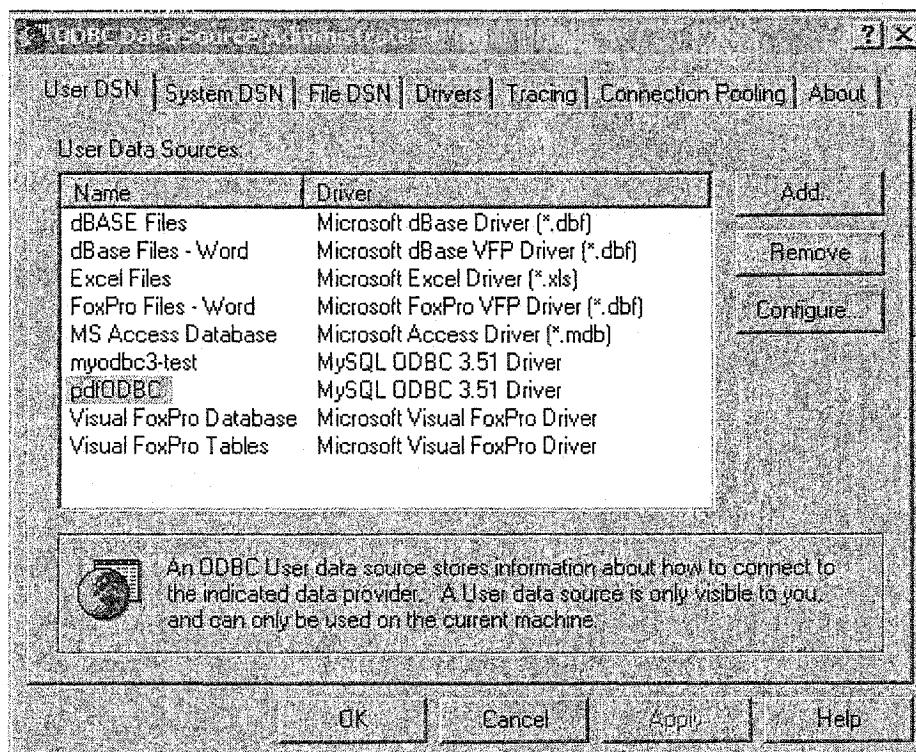


Figure 3.4 User Data Source in Control Panel

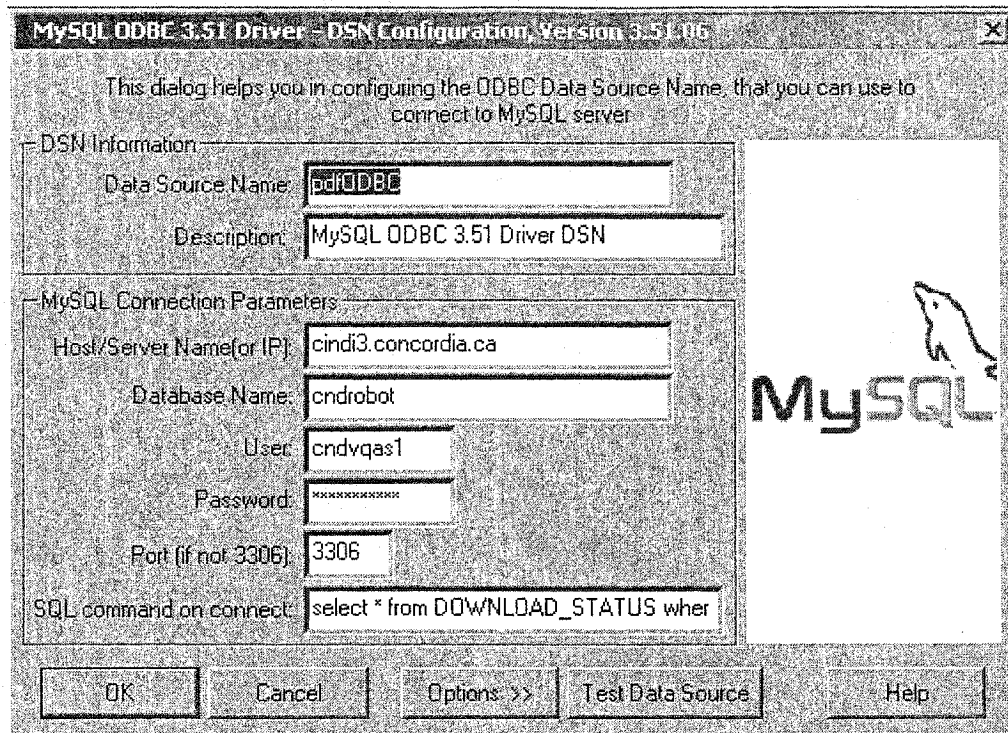


Figure 3.5 Configuration of MyODBC

Upon clicking OK, the Data Sources dialog box appears, and the ODBC Administrator updates the registry information. The typed user name and connect string become the default connection values for this data source. To make sure the connection between API and MySQL database is correct, it is necessary to test by utilizing the button Test Data Source. The successful connection returns the message shown in Figure 3.6.

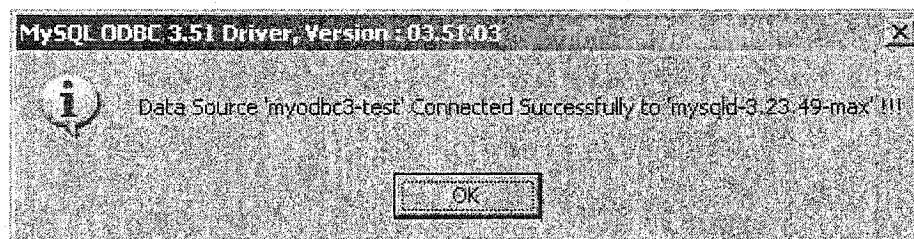


Figure 3.6 Test Message of MyODBC

After successfully connecting to MySQL database, FCS can access the CINDI database on the MySQL server using the information saved in MyODBC data source.

As mentioned in section 3.3.2, the cooperation of Omniformat and Pdf995 was chosen as the converter in FCS. Therefore, Pdf995 and Omniformat were installed on the Windows platform. The working principle of the converter will be described in the next section.

3.3.4 Design of FCS

The purpose of FCS is to provide a single document format to facilitate document processing in the subsequent subsystems of CINDI. Since PDF is emerging as the current favorite format for electronic documents, it was chosen as the single format for CINDI. Therefore, non-PDF documents such as TXT, PS, WPD, HTML, DOC, and LaTeX files located and downloaded by CINDI Robot need to be converted into PDF format.

Based on the proposed solution, FCS was developed as an automatic file conversion system. FCS checks the CINDI Robot database on the Linux platform every 30 minutes. When new downloaded non-PDF documents are found in the database, it securely and automatically transfers these documents from the non-PDF repository on the Linux platform to the repository on the Windows platform for conversion. Moreover, FCS employs Pdf995 and Omniformat to convert the downloaded documents into PDF files.

After conversion, FCS sends the PDF version of the documents back to the temporary repository on the Linux platform for filtering and writes information about PDF file name, conversion flag, conversion date, and file location into the CINDI database. Based on the

functionality, FCS is decomposed into three components: FCS Repository, FCSd, as well as Converter. The architecture of FCS is given in Figure 3.7.

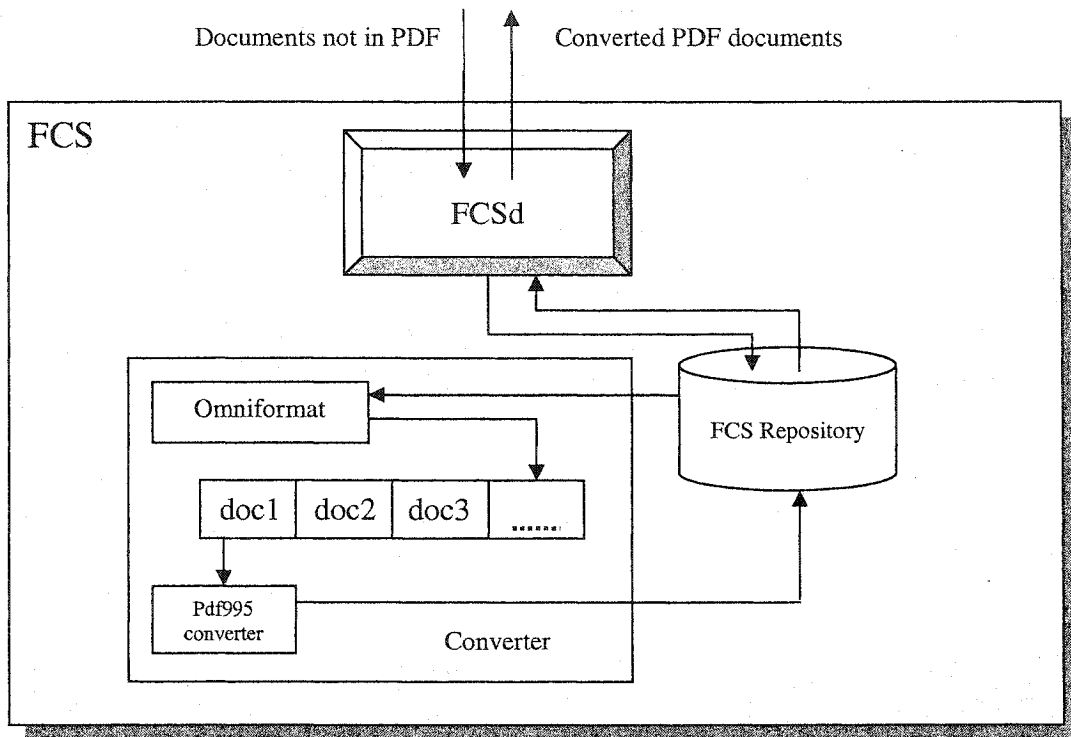


Figure 3.7 Architecture of File Converting System

From Figure 3.7, FCS Repository is a repository on the Windows platform to store the downloaded non-pdf documents. FCSd is responsible for the automatic file transfer between two platforms, while the Converter is responsible for the automatic file conversion. The Converter is composed of Pdf995 and Omniformat. Pdf995 is a PDF converter, while Omniformat is a background process to find new non-pdf documents in FCS Repository and send them to the converting queue for conversion by Pdf995.

The process starts with FCSd monitoring the CINDI Robot database. If non-PDF documents are found, FCSd transfers them into the FCS Repository on the Windows platform. Once non-PDF documents are downloaded into the FCS Repository,

Omniformat puts them in the converting queue. Next, Pdf995 obtains documents from the queue and converts them into PDF according to the FIFO priority. The converted PDF files are returned to the FCS Repository and the original files are deleted. After all non-PDF documents are converted into PDF, FCSd sends them back to the temporary repository on the Linux platform. Simultaneously, FCSd writes converting information about conversion flag, conversion date, file name, and file location in the CINDI Robot database. Once all converted documents are transferred back to the Linux platform, a cycle of FCSd is finished. Another cycle will be executed after 30 minutes and FCSd “sleeps” during this time.

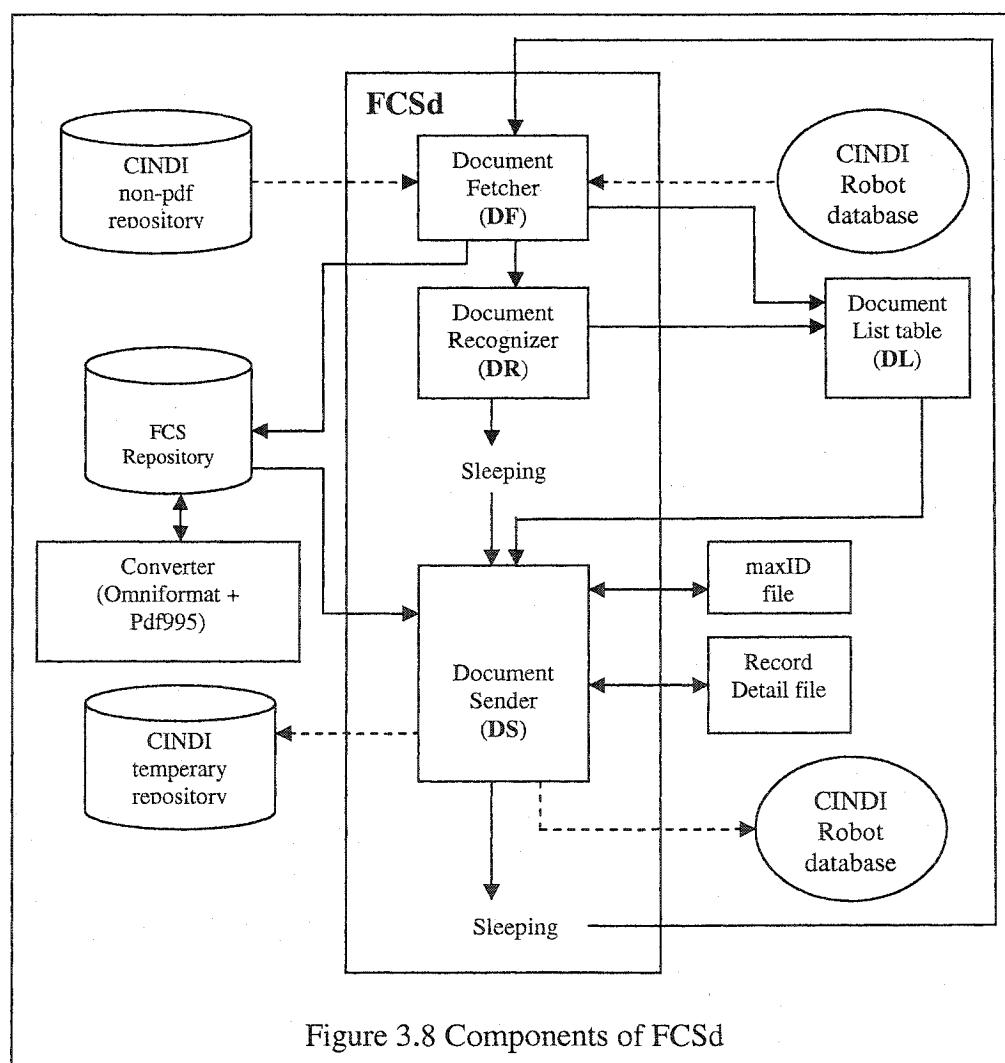
3.3.5 Implementation of FCSd

FCSd is a program that integrates FCS into CINDI system. It has the function of securely and automatically transferring files between the Windows platform and the Linux platform. FCSd performs one cycle of jobs every 30 minutes. The detail implementation of FCSd is given below.

Initially, FCSd checks non-PDF documents in the DOWNLOAD_STATUS table in the CINDI Robot database. If non-PDF files are found, it transfers batch of the documents to the FCS Repository, and then goes to sleep to wait for file conversions. The average PDF conversion time for a document, using a set of 300 files, was found to be about 6 seconds. Therefore, account for the variation of the conversion time, FCSd allows 7 seconds for each file conversion during which time it is put in a “sleep” status. When FCSd wakes up, it searches the converted PDF documents in the FCS Repository, sends them back to the temporary repository on the Linux platform for filtering, and then FCSd deletes the

transferred PDF files from the FCS Repository. Finally, FCSd sets the conversion flag in the DOWNLOAD_STATUS table to "TRUE" and writes a record about file name, conversion date, and location for each file in the CONVERT_PDF table. Eventually, FCSd finishes this cycle and goes to sleep for 30 minutes. The ID in the CONVERT_PDF table is updated by FCSd. To make sure FCSd sets correct IDs for converted documents in the next cycle, the maximum ID used for CONVERT_PDF in this cycle is stored in a file (maxID file).

FCSd consists of three main components: Document Fetcher (DF), Document Recognizer (DR), and Document Sender (DS). The components of FCSd are shown in Figure 3.8.



As depicted in Figure 3.8, DF is responsible for four tasks. When a new cycle starts, DF initializes a Document List (DL) table to be used by DS. Then, it connects to the Robot database on the CINDI MySQL server and searches for non-PDF documents in the database. Lastly, DF fetches these documents from the location stored in the DOWNLOAD_STATUS table and securely transfers them to the FCS Repository using pscp.

DR stores the file name and the ID for each fetched document in the DL table. Later DS will retrieve this information and save it in the CONVERT_PDF table. Also, DR saves two file names with extension of pdf into the DL table. One file name has the same case as the original name's prefix, while the other is in lower case. The purpose of the first pdf name is to store the prefix of the original file name prefix given by the CINDI Robot in CONVERT_PDF, while the file name in lower case is utilized by DS to match the converted file name in the FCS Repository. This is because Pdf995 converter is file name insensitive, and the converted documents are named in lower case. For example, the document "ABc.html" will be stored as "abc.pdf" after conversion. By matching the prefix of the lower case name in the DL table with the converted PDF name, the original file name with pdf extension is saved in the CONVERT_PDF table. The sample content of the DL table is shown in Table 3.2

Table 3.2 Samples of Document List

Original file name	ID in DOWNLOAD_STATUS	Converted pdf file name in lower case	Original file name with PDF extension
THESIS.doc	2655	thesis.pdf	THESIS.pdf
advice-to-grads-fac.pdf	2602	advice-to-grads-fac.pdf	advice-to-grads-fac.pdf
68.html	1264	68.pdf	68.pdf
CS-TR-4527.pdf	1594	cs-tr-4527.pdf	CS-TR-4527.pdf
.			
.			
.			

As shown in Table 3.2, thesis.pdf is matched with THESIS.doc to rename it as THESIS.pdf which is saved in CONVERT_PDF table.

After DR records the details for all fetched documents in the DL table, FCSd goes to sleep. When the Converter finds the downloaded documents in the FCS Repository, it begins to convert them into PDF.

When FCSd wakes up, DS prepares to transfer the converted documents back to the temporary repository on the Linux platform, described as follows. First, DS looks for the maxID file where the maximum ID in the CONVERT_PDF table is stored. If the maxID file does not exist, which means the CONVERT_PDF table is empty, then DS will create one and set the value to 0. The purpose of remembering the maximum ID is to make the IDs in the CONVERTED_PDF table consecutive. Then, DS creates Record Detail file containing a list of file name, file size, and conversion date and time for documents in the FCS Repository. Next, DS takes the PDF documents listed in the Record Detail file from the FCS Repository and sends them back to the temporary repository on the Linux platform for filtering by DFS. The documents not converted at that time would be sent in next cycle. Next, DS sets the PDF conversion flag in the DOWNLOAD_STATUS table to 1 and stores conversion information in the CONVERT_PDF table. In order to save the original file name in the CONVERT_PDF table, DS looks up the DL table using the converted file name. The original file name with PDF extension for the matched file in the DL table is stored in the CONVERT_PDF table together with conversion date and the file location in CINDI. To meet the requirement of SQL statement in MySQL, some data

types need to be processed. For example, date format needs to be changed from “MM/DD/YY” to “YYYY-MM-DD”, the ID in the DL table needs to be cast from integer to character string. Finally, DS locally saves the maximum ID from the CONVERT_PDF table in the maxID file to be used in the next cycle. FCSd is then suspended for thirty minutes until the next cycle.

FCSd is written in C++ and runs in the Windows environment. The overall algorithm of FCSd is given below.

Input: non-PDF documents from the CINDI Robot System

Output: the converted PDF documents

Begin

while true **do**

begin

 initialize an empty Document List table DL;

 DF connects to the CINDI Robot database d;

 DF selects a non-PDF document set S from d;

 DF stores S in local as s;

while not end of s **do**

begin

 DR takes a file f from s;

 DR transfers f from the location displayed in s to the FCS Repository D;

 DR cast the ID in s from integer to character string as docID;

 DR tokenizes the prefix(fnf) of the file name in s as fn;

 DR appends extension pdf to fnf as fn_pdf_uc;

 DR changes fnf into lower case and append extension pdf to it as fn_pdf_lc;

 DR stores fn, docID, fn_pdf_uc, and fn_pdf_lc into DL;

End

 sleep for 8 seconds for each file in s;

If there is no maxID file **then**

 DS creates ID file;

 DS sets ID as 0;

 DS gets maximum ID from the maxID file;

 DS creates Record Detail file I to store information for the files in the FCS Repository

while not end of I **do**

begin

 DS gets one PDF file f from I;

 DS searches table DL for f;

If fn_pdf_lc==f **then**

 DS gets fn_pdf_uc;

 DS gets docID;

 DS gets date from I;

 DS sets location;

 DS sends f to the temporary repository in the Linux Platform;

 DS updates pdf_flag in DOWNLOAD_STATUS to 1 ;

next cycle starts, the DL table is automatically initialized as an empty table by pointing the “head_ptr” and the “tail_ptr” to NULL.

3.4 Document Filtering Subsystem (DFS)

3.4.1 The Approach

The function of DFS is to filter out irrelevant information from the documents retrieved by the CINDI Robot. As introduced in Chapter 2, typical documents on the Web are e-mails, news items, personal home pages, resumes, letters, group discussions, graphics, and academic documents. Since the CINDI system is a digital library in the academic area, scientific documents including theses, technical reports, academic papers, and FAQs are the genre of documents in CINDI. Documents such as e-mails, resumes, group discussions, letters, and other similar documents are considered irrelevant. Hence, the set of documents downloaded from the Web by the CINDI Robot is processed by DFS into two subsets: *accepted* and *rejected* as described in the following formulas; the former set contains documents which are accepted and the latter set contains files which are rejected.

$$downloads = accepted \cup rejected$$

$$accepted = \{f : | f \in downloads \wedge (f \in thesis \vee f \in report \vee f \in paper \vee f \in faq) \}$$

$$rejected = \{f : | f \in downloads \wedge (f \notin thesis \wedge f \notin report \wedge f \notin paper \wedge f \notin faq) \}$$

If DFS finds a document $d \in accepted$, then d will be accepted; in contrast, if $d \notin accepted$, then DFS will reject this document and put it into the CINDI trash.

DFS makes a filtering decision according to the document structure by comparing the candidate document with the predefined DTDs to determine the types of acceptable

documents. Each element in the DTDs has certain weight. The weight assigned to a candidate document depends on the presence of the DTD elements in this document. The higher the weight, the closer the document is to the predefined document type.

3.4.2 Document Type Definition (DTD)

The purpose of a DTD is to define the legal building blocks of a document. It defines the document structure with a list of legal elements. The building blocks of a semi-structured document are made up of elements, tags, attributes, entities, Parsed Character Data (PCDATA), and Character Data (CDATA). CDATA is text that will not be parsed by a parser. Elements are the main building blocks of a document; they can contain text, other elements, or be empty. For elements in a DTD order matters. Tags are used to markup elements. Attributes provide extra information about elements; they are placed inside the starting tag of an element and come in name and value pairs. Entities are the variables used to define common text. Tags inside the text will not be treated as markup and entities will not be expanded. Since the source documents of DFS are text files, we define DTDs using elements, tags and CDATA building blocks. The question mark “?” follows an optional element. The plus sign “+” after an element means that there should be at least one such element. The keyword `#REQUIRED` is used to define an element that has to be present in its parent element; the keyword `#IMPLIED` defines an element that may be included in its parent element. For DFS, an element has higher weight than an implied element if it has `#REQUIRED` key words and none of its ancestors are optional. In addition to the acceptable document types, the DTDs for documents which are not acceptable for CINDI are also defined.

3.4.2.1 DTD for Theses

The DTD for theses consists of three main elements: front matter, body, and back matter. The front matter has a title page, certificate of approval (certifapproval), acknowledgments, abstract, table of contents (toc), list of tables, list of figures, and list of symbol abbreviation. On the title page, the elements title, author, terms of submission, and copyright are included. Since some theses include sentences such as “A Thesis in Department of Computer Science” on the title page, the appearance of the string “Thesis” is also defined as an element of DTD. The element “abstract” is defined as having the keyword “Abstract” and its contents. “acknowledgment” is defined in the same way. Table of contents, list of tables, and list of figures are in the form of ordered lists. Each item in the list ends with a page number.

The body in the DTD of theses is its primary content. By common sense, some documents cannot be considered as theses or technical reports. For example, we do not consider a document that has only one chapter or has more than one chapter but most of the chapters have at most one section as a thesis. Therefore, we specify that the body structure of a thesis must have more than three chapters and each chapter must have at least two sections. The chapter of a thesis may have one or more paragraphs before the first section starts. In each section, there are subsections or paragraphs. Chapter title, section title, and subsection title are included in the elements of chapter, section, and subsection respectively. A paragraph should be defined on sentence level. Due to the fact that analyzing the sentences of the converted text files is very complex and unreliable, we define the text between two empty lines as a paragraph. The text should have more than 3

lines whose string lengths are more than 50 (This is the average line length in the converted text files of 50 test documents).

The element “back” consists of all the matter following the text proper; it includes the bibliography (bib), references, and appendix, if any. The element bib consists of the term “bibliography” (bibword) and its contents. Since the bibliography can be recognized through checking the identifier such as “[“, the element bibword is defined as #IMPLIED. The element “references” is similar to “bib”. In contrast to the element “bib”, without the term “Appendix” (appword), an appendix is difficult to be identified. Therefore, the element “appword” is #REQUIRED. All the “back” elements are defined as #IMPLIED. The DTD for theses is presented below.

thesis.dtd

```
<?xml version="1.0" >
<!ELEMENT thesis ( front, body, back? ) >
<!ELEMENT front (#CDATA titlepage, copyright?, certifapproval, acknowledgments?,
abstract, toc, tablelist?, figurelist?, symbolabbrevlist? ) >
<!ELEMENT titlepage ( title | author | thesis? | submission? | copyright? ) >
<!ELEMENT title ( #CDATA ) #REQUIRED >
<!ELEMENT author ( #CDATA ) #REQUIRED >
<!ELEMENT thesis ( #CDATA ) #IMPLIED >
<!ELEMENT submission ( #CDATA ) #IMPLIED >
<!ELEMENT copyright ( #CDATA ) #IMPLIED >
<!ELEMENT certifapproval ( #CDATA ) #IMPLIED >
<!ELEMENT acknowledgments (#CDATA ackword | ackparagraph* ) >
<!ELEMENT ackword ( #CDATA ) #REQUIRED >
<!ELEMENT ackparagraph ( #CDATA ) #IMPLIED >
<!ELEMENT abstract (#CDATA absword | absparagraph* ) >
<!ELEMENT absword ( #CDATA ) #REQUIRED >
<!ELEMENT absparagraph ( #CDATA ) #REQUIRED >
<!ELEMENT toc (#CDATA tocword | tocontents) >
<!ELEMENT tocword ( #CDATA ) #REQUIRED >
<!ELEMENT tocontents ( #CDATA ) #REQUIRED >
<!ELEMENT tablelist (#CDATA tlword | tlcontents) >
<!ELEMENT tlword ( #CDATA ) #REQUIRED >
<!ELEMENT tlcontents ( #CDATA ) #REQUIRED >
<!ELEMENT figurelist (#CDATA flword | flcontents) >
<!ELEMENT flword ( #CDATA ) #REQUIRED >
<!ELEMENT flcontents ( #CDATA ) #REQUIRED >
<!ELEMENT symbolabbrevlist (#CDATA abbrword | abbrcontents) >
```

```

<ELEMENT abbrword ( #CDATA ) #REQUIRED >
<ELEMENT abbrcontents ( #CDATA ) #REQUIRED >
<ELEMENT preface ( #CDATA preword | precontents ) >
<ELEMENT preword ( #CDATA ) #REQUIRED >
<ELEMENT precontents ( #CDATA ) #REQUIRED >

<ELEMENT body ( #CDATA chapter+ ) >
<ELEMENT chapter ( #CDATA chtitle | section | chparagraph? ) >
<ELEMENT chtitle ( #CDATA ) #REQUIRED >
<ELEMENT section ( #CDATA sectitle | subsection? | secparagraph? ) >
<ELEMENT sectitle ( #CDATA ) #REQUIRED >
<ELEMENT subsection ( #CDATA subsectitle | subsecparagraph+ ) >
<ELEMENT subsectitle ( #CDATA ) #REQUIRED >
<ELEMENT subsecparagraph ( #CDATA ) #REQUIRED >
<ELEMENT secparagraph ( #CDATA ) #REQUIRED >
<ELEMENT chparagraph ( #CDATA ) #REQUIRED >

<ELEMENT back ( #CDATA bib? | references? | appendix? ) >
<ELEMENT bib ( #CDATA bibword? | bibcontents ) >
<ELEMENT bibword ( #CDATA ) #IMPLIED >
<ELEMENT bibcontents ( #CDATA ) #REQUIRED >
<ELEMENT ref ( #CDATA refword? | refcontents ) >
<ELEMENT refword ( #CDATA ) #IMPLIED >
<ELEMENT refcontents ( #CDATA ) #REQUIRED >
<ELEMENT appendix ( #CDATA appword | appcontents ) >
<ELEMENT appword ( #CDATA ) #REQUIRED >
<ELEMENT appcontents ( #CDATA ) #REQUIRED >

```

3.4.2.2 DTD for Technical Reports

The structure of a standard technical report is similar to a thesis. Although the technical report does not always have abstract, acknowledgment, bibliography, and appendix, they are still included in the DTD of technical reports to avoid the loss of acceptable documents. The appearance of the term “Report” on the front page is also defined in the DTD. The DTD for technical reports is shown below.

report.dtd

```

<?xml version="1.0" >
<ELEMENT report ( front, body, back? ) >
<ELEMENT front ( #CDATA titlepage, acknowledgments?, abstract?, toc, tablelist?, figurelist? ) >
<ELEMENT titlepage ( #CDATA title | author* | report? | copyright? ) >
<ELEMENT title ( #CDATA ) #REQUIRED >
<ELEMENT author* ( #CDATA ) #IMPLIED >
<ELEMENT report ( #CDATA ) #IMPLIED >
<ELEMENT copyright ( #CDATA ) #IMPLIED >
<ELEMENT acknowledgments ( #CDATA ackword | ackparagraph+ ) >

```



```

<!ELEMENT ackword ( #CDATA ) #REQUIRED >
<!ELEMENT ackparagraph ( #CDATA ) #REQUIRED >
<!ELEMENT abstract ( #CDATA absword | absparagraph+ ) >
<!ELEMENT absword ( #CDATA ) #REQUIRED >
<!ELEMENT absparagraph ( #CDATA ) #REQUIRED >
<!ELEMENT toc ( #CDATA tocword | tocontents ) >
<!ELEMENT tocword ( #CDATA ) #REQUIRED >
<!ELEMENT tocontents ( #CDATA ) #REQUIRED >
<!ELEMENT tablelist ( #CDATA tlword | ticontents ) >
<!ELEMENT tlword ( #CDATA ) #REQUIRED >
<!ELEMENT ticontents ( #CDATA ) #REQUIRED >
<!ELEMENT figurelist ( #CDATA flword | flcontents ) >
<!ELEMENT flword ( #CDATA ) #REQUIRED >
<!ELEMENT flcontents ( #CDATA ) #REQUIRED >
<!ELEMENT preface ( #CDATA preword | precontents ) >
<!ELEMENT preword ( #CDATA ) #REQUIRED >
<!ELEMENT precontents ( #CDATA ) #REQUIRED >

<!ELEMENT body ( #CDATA chapter+ ) >
<!ELEMENT chapter ( #CDATA chtitle | section | chparagraph? ) >
<!ELEMENT chtitle ( #CDATA ) #REQUIRED >
<!ELEMENT section ( #CDATA sectitle | subsection? | secparagraph? ) >
<!ELEMENT sectitle ( #CDATA ) #REQUIRED >
<!ELEMENT subsection ( #CDATA subsectitle | subsecparagraph+ ) >
<!ELEMENT subsectitle ( #CDATA ) #REQUIRED >
<!ELEMENT subsecparagraph ( #CDATA ) #REQUIRED >
<!ELEMENT secparagraph ( #CDATA ) #REQUIRED >
<!ELEMENT chparagraph ( #CDATA ) #REQUIRED >

<!ELEMENT back ( #CDATA bib? | references? | appendix? ) >
<!ELEMENT bib ( #CDATA bibword? | bibcontents ) >
<!ELEMENT bibword ( #CDATA ) #IMPLIED >
<!ELEMENT bibcontents ( #CDATA ) #REQUIRED >
<!ELEMENT references ( #CDATA refword? | refcontents ) >
<!ELEMENT refword ( #CDATA ) #IMPLIED >
<!ELEMENT refcontents ( #CDATA ) #REQUIRED >
<!ELEMENT appendix ( #CDATA appword | appcontents ) >
<!ELEMENT appword ( #CDATA ) #REQUIRED >
<!ELEMENT appcontents ( #CDATA ) #REQUIRED >

```

* we look for a “slot” of author and not trying to identity one or more authors.

3.4.2.3 DTD for FAQs

The structure of FAQs is generally composed of a series of questions and the corresponding answers. A document with solely questions cannot be considered as a FAQ. Thus, paragraph is also a required key element in the element “answer” as described below.

faq.dtd

```
<?xml version="1.0" >
<!ELEMENT faq ( title, content ) >
<!ELEMENT title (#CDATA FAQ ) >
<!ELEMENT FAQ (#CDATA ) #REQUIRED >
<!ELEMENT content (#CDATA (question, answer)+ ) >
<!ELEMENT question (#CDATA queword? | quesentence) >
<!ELEMENT queword (#CDATA ) #IMPLIED >
<!ELEMENT quesentence (#CDATA ) #REQUIRED >
<!ELEMENT answer (#CDATA answerword? | paragraph+ ) >
<!ELEMENT answerword ( #CDATA ) #IMPLIED >
<!ELEMENT paragraph (#CDATA ) #REQUIRED >
```

The DTD for FAQs consists of title and contents. The term FAQ is defined in the title.

The content of FAQ is made up of many pairs of question and answer. One question may begin with the term “question”; it must end with a question mark “?”. The element answer may start with the term “answer”; it should include at least one paragraph.

3.4.2.4 DTD for Academic Papers

The structure of an academic paper is different from any of the document types above. Since different institutes and individuals use different formats of elements for the same paper, the DTD for academic papers can only identify the main characteristics of technical and scientific papers. The main feature of an academic paper is having a title and body structure consisting of sections, subsections, and paragraphs. The DTD for academic papers is defined as follows.

paper.dtd

```
<?xml version="1.0" >
<!ELEMENT paper ( front, body, back? ) >
<!ELEMENT front (#CDATA title | abstract? | keywords? | contents) >
<!ELEMENT title ( #CDATA ) #REQUIRED >
<!ELEMENT abstract (#CDATA absword? | absparagraph+ ) >
<!ELEMENT absword( #CDATA ) #IMPLIED >
<!ELEMENT absparagraph ( #CDATA ) #REQUIRED >
<!ELEMENT keywords ( #CDATA keyword | keylist+ ) >
<!ELEMENT keyword ( #CDATA ) #IMPLIED >
<!ELEMENT keylist ( #CDATA ) #REQUIRED >
```

```

<!ELEMENT body ( #CDATA section* ) >
<!ELEMENT section ( #CDATA sectitle | subsection? | secparagraph* ) >
<!ELEMENT sectitle ( #CDATA ) #REQUIRED >
<!ELEMENT subsection ( #CDATA subsectitle | subsecparagraph* ) >
<!ELEMENT subsectitle ( #CDATA ) #REQUIRED >
<!ELEMENT subsecparagraph ( #CDATA ) #REQUIRED >
<!ELEMENT secparagraph ( #CDATA ) #REQUIRED >

<!ELEMENT back ( #CDATA ack? | bib? | references? | appendix? ) >
<!ELEMENT ack ( #CDATA ackword | ackcontents ) >
<!ELEMENT ackword ( #CDATA ) #REQUIRED >
<!ELEMENT ackcontents ( #CDATA ) #REQUIRED >
<!ELEMENT bib ( #CDATA bibword? | bibcontents ) >
<!ELEMENT bibword ( #CDATA ) #IMPLIED >
<!ELEMENT bibcontents ( #CDATA ) #REQUIRED >
<!ELEMENT references ( #CDATA refword? | refcontents ) >
<!ELEMENT refword ( #CDATA ) #IMPLIED >
<!ELEMENT refcontents ( #CDATA ) #REQUIRED >
<!ELEMENT appendix ( #CDATA appword | appcontents ) >
<!ELEMENT appword ( #CDATA ) #REQUIRED >
<!ELEMENT appcontents ( #CDATA ) #REQUIRED >

```

The DTD for academic papers contains three parts: front, body, and back. The back part is optional. The front matter has title, abstract, keywords, and contents. Since the term “abstract” is not denoted explicitly in some papers, the element absword is defined as optional. Unlike the author of a thesis who is only one person who is listed immediately after title, the author in an academic paper may be one or more persons who are listed in one of several ways. For example, the three authors A, B, and C may appear in the form of “A B C”, “A B & C”, “A;B;C”, “A, B, and C”, or “A & B & C”. Sometimes multiple authors are followed by their address. The identification of multiple authors is more complex and time consuming than for single author. However, author is not a major concern in deciding if a document is an academic paper or not. Thus, the DTD for academic papers does not define author as an element. Occasionally, table of contents is also included in such papers. In order to preserve all academic papers, table of contents is defined to be an implied element.

The body of the DTD for academic papers includes more than one section. Each section should consist of a subsection or paragraphs. The back page is composed of acknowledgments (ack), bibliography, and appendix. Most of the time, the element ack appears in the back of a paper; thus, it is defined in the back of the DTD for academic papers.

3.4.2.5 DTD for E-mails

An e-mail is composed of two parts: header and body. Header consists of date, from, to, and subject elements of which date, from, and to are required. Considering that the subject is often omitted by the writer in an e-mail, the element subject is defined as #IMPLIED. The body of an e-mail contains the salutation to the receiver (begining), the primary contents (paragraph), and the end matter (ending) which include the ending words and the name of the sender.

e-mail.dtd:

```
<?xml version = "1.0">
<!ELEMENT email (header, body )>
<!ELEMENT header (#CDATA date | from | to | subject )>
<!ELEMENT date (CDATA) #REQUIRED>
<!ELEMENT from (CDATA) #REQUIRED>
<!ELEMENT to (CDATA) #REQUIRED>
<!ELEMENT subject (CDATA) #IMPLIED>
<!ELEMENT body (#CDATA begining | paragraph+ | ending ) >
<!ELEMENT begining (CDATA) #REQUIRED>
<!ELEMENT paragraph (CDATA) #REQUIRED>
<!ELEMENT ending (CDATA) #REQUIRED>
```

3.4.2.6 DTD for Resumes

The DTD for resumes includes personal information, skills, experience, education, personal portrait, and other elements as shown below.

resume.dtd

```
<?xml version = "1.0">
<!ELEMENT resume (header, objective, skills, language?, experience, education,
accomplishment?, personal traits?, academic project?, Activities?)
<!ELEMENT header (#CDATA name | address | email | phone | date) >
<!ELEMENT name (#CDATA) #REQUIRED>
<!ELEMENT address (#CDATA) #REQUIRED>
<!ELEMENT email (#CDATA) #IMPLIED >
<!ELEMENT phone (#CDATA) #REQUIRED>
<!ELEMENT date (#CDATA) #IMPLIED >
<!ELEMENT objective (#CDATA) #REQUIRED>
<!ELEMENT skills (#CDATA) #REQUIRED >
<!ELEMENT language (#CDATA) #IMPLIED >
<!ELEMENT experience ((#CDATA employer | from | to | responsibility | description) >
<!ELEMENT employer (#CDATA) #REQUIRED >
<!ELEMENT from (#CDATA) #REQUIRED >
<!ELEMENT to (#CDATA) #REQUIRED >
<!ELEMENT responsibility (#CDATA) #REQUIRED >
<!ELEMENT description (#CDATA) #IMPLIED >
<!ELEMENT education (#CDATA) #REQUIRED >
<!ELEMENT accomplishment (#CDATA) #IMPLIED >
<!ELEMENT personal traits (#CDATA) #IMPLIED >
<!ELEMENT academic project (#CDATA) #IMPLIED >
<!ELEMENT activities(#CDATA) #IMPLIED >
```

3.4.2.7 DTD for Letters

A letter should have contact information and content. Therefore, the DTD for letters includes at least one contact person, contact address, the destination of the letter, and at least one paragraph in the body.

letter.dtd

```
<?xml version = "1.0">
<!ELEMENT letter (contact+, body)>
<!ELEMENT contact ((#CDATA name | address+ )>
<!ELEMENT name (#CDATA) #REQUIRED>
<!ELEMENT address (#CDATA toname | city | province | country? | zip code | e-mail? | phone? | fax? )>
<!ELEMENT toname (#CDATA) #REQUIRED>
<!ELEMENT city (#CDATA) #REQUIRED>
<!ELEMENT province (#CDATA) #REQUIRED>
<!ELEMENT country (#CDATA) #REQUIRED>
<!ELEMENT zip code (#CDATA) #REQUIRED>
<!ELEMENT e-mail (#CDATA) #IMPLIED>
<!ELEMENT phone (#CDATA) #IMPLIED>
<!ELEMENT fax (#CDATA) #IMPLIED>
<!ELEMENT body (#CDATA begin | paragraph+ | end ) >
<!ELEMENT begin (CDATA) #REQUIRED>
<!ELEMENT paragraph (CDATA) #REQUIRED>
<!ELEMENT end (CDATA) #REQUIRED>
```

3.4.2.8 DTD for Discussion Groups

A typical discussion group gives the source of the message (from), the date, subject, newsgroup, content, and attachment. A discussion group may also contain threads, links to next message and previous message, receiver, newsgroup, attachment, and sorting method. In the sort option, start and end date and time, author, subject index, and thread index elements are included. Based on the above information, the DTD for discussion groups is defined as:

discussiongroup.dtd

```
<?xml version = "1.0">
<ELEMENT discussiongroup (from+, date+, thread? next message?, previous message?,
subject+, newsgroup?, receiver?, attachment?, paragraph+, sort? )>
<ELEMENT from (#CDATA) #REQUIRED>
<ELEMENT date (#CDATA) #REQUIRED>
<ELEMENT thread (#CDATA) #IMPLIED>
<ELEMENT next message (#CDATA) #IMPLIED >
<ELEMENT previous message (CDATA) #IMPLIED >
<ELEMENT subject (#CDATA) #REQUIRED>
<ELEMENT receiver (#CDATA) #REQUIRED >
<ELEMENT attachment (#CDATA) #IMPLIED>
<ELEMENT paragraph (#CDATA) #REQUIRED >
<ELEMENT sort (#CDATA start? | end? | date? | author? | subject index? | thread index? ) >
<ELEMENT start (#CDATA) #IMPLIED>
<ELEMENT end (#CDATA) #IMPLIED>
<ELEMENT date (#CDATA) #IMPLIED>
<ELEMENT author (#CDATA) #IMPLIED>
<ELEMENT subject (#CDATA) #IMPLIED>
<ELEMENT thread index (#CDATA) #IMPLIED>
```

3.4.3 Implementation of DFS

Documents in CINDI are maintained in PDF format. Since PDF files present text and graphics in Adobe imaging model, they are composed of non-displayable characters. In order to extract the structure from a PDF document, the PDF file needs to be converted to a plain text file. Therefore, the converter pdftotext from the open source package Xpdf is employed to convert PDF documents into plain text files. After conversion, some

formatting of PDF documents such as font, font style, font size, and page layout are lost. However, using what may be an incomplete text structure, DFS must extract a structure in order to determine if a document is acceptable or not. DFS is implemented using C/C++ and runs on a Linux platform.

3.4.3.1 Thesis/Report Structure Extraction

A thesis in original layout generally consists of many blocks of text (a block of text is a piece of text separated from the other text by at least one empty line before and after it) such as title block, author block, abstract block, acknowledgment block, section block, and paragraph block. In a PDF file, all these blocks are arranged in the order of front, body, and back. These features are basically preserved in the converted text file.

The front page of a standard thesis often includes blocks for title, author, terms of submission, copyright, certificate of approval, acknowledgments, abstract, table of contents, list of tables, and list of figures. Title is the first sentence in the first block; except for number and stop words, each term in a title begins with a capital letter. Author is in the second block. Sometimes there is a phrase such as “A Dissertation Presented By” or the term “by” before the author’s name. Some elements such as “Acknowledgment” appears in one line in a document and are immediately followed by their content or after several empty lines. Abstract is similar to acknowledgments but, in addition, the abstract often has an additional block for the author information between the term “Abstract” and its content. Table of contents is in the form of a list. Each item in the list begins with the term “chapter” or a number and ends with a page number. Sometimes one long item

occupies several lines. A similar structure is applied for list of tables and list of figures.

The front matter of a technical report has a similar structure to the front matter of a thesis.

According to the defined DTDs, the body of a thesis or a report consists of chapters. Chapters are composed of chapter titles, sections, and paragraphs; sections include section title and may be made up of subsections. Chapter title, section title, and subsection title are a sentence or a phrase, which consists of terms beginning with capital letters except for stop words and numbers. These titles often begin with numbers in the form of “1”, “1.2”, or “1.2.3”. Chapter title number also has the variation “Chapter 1”. After conversion by pdftotext, the following problems have to be addressed.

- At times there is no space between the section number and the section title. For example, “1Introduction” in the converted text file was “1 Introduction” in the original PDF document.
- Figures and tables in a PDF file cannot be correctly converted. Only labels are left in the converted text file. Labels for a figure or a table are set into one block, line by line. There are often empty lines between this block and the text before and after it.
- An academic document often have numbered lists. After conversion, the list may have the same format as a section title or subsection title.
- Formulas in converted text are similar to section titles, subsection titles, or sentences. Continuous formulas often occupy one or several blocks in the converted text file.

The back page of a thesis or a technical report contains bibliography, references, and appendix. Bibliography and references are citation lists. The term “Bibliography” or “References” is usually a single line and in front of the list. Each item in the list begins with the author name, a number, the identifier “[“ or “{“.

Thesis/report structure extraction is a module of DFS. Since theses and technical reports have similar DTD and the DTD of technical reports is a subset of the DTD of theses, a candidate document is matched with the two DTDs in the same algorithm. The differences between them lie in the elements and weight assignments. To extract the thesis-like or report-like document structure from a candidate document, DFS divides this document into three parts: front, body, and back, according to the identifiers. The identifier for the beginning of body structure for theses is:

- A string beginning with a number in the form of “1. “ or a term “Chapter”.
- Each term in the string should begin with a capital letter except for numbers and stopwords.
- The string should not be a formula, a label, or an item in a list.

The identifier for the beginning of back part for theses is the appearance of the back element “References”, “Bibliography”, or “Appendix”.

DFS puts the front part, each section or subsection (if any) of the body, and the back part sequentially into a block of buffers. DFS scans each block of buffers back and forth to search for elements in the DTD. If DFS finds one element of a thesis, it assigns thesis weight to this document; if it finds an element for both thesis and technical report, then it assigns both thesis weight and technical report weight. When searching for paragraphs in

the body, DFS ignores the text blocks for labels and formulas between and in the paragraphs. If the thesis weight or technical report weight is high enough, the module of thesis/report matching will return 1 or 2, which means DFS will accept this document as a thesis or a technical report; otherwise, it will return 0 to throw this document into the CINDI trash as shown in the following algorithm.

Input: a converted text document d_{txt} from temporary repository

Output: 1 if d_{txt} is a thesis
 2 if d_{txt} is not a technical report
 0 if d_{txt} is not a thesis or a report

```

begin
  initial body structure;
  open  $d_{\text{txt}}$ ;
  //processing for front
  pass empty lines;
  ignore page number line;
  ignore date line;
  get one line  $l_1$  from  $d_{\text{txt}}$ ;
  if all terms except for stop words and numbers in  $l_1$  begin with capital letter then
     $l_1$  is the title of  $d_{\text{txt}}$ ;
    update 2 to thesis weight wt;
    update 2 to thesis weight wr;
  pass empty lines;

  initial a block of buffers  $b_1$ ;
  while (!end of  $d_{\text{txt}}$  and body not starts and back not starts)
    begin
      get one line  $l_2$  from  $d_{\text{txt}}$ ;
      put  $l_2$  into one buffer of  $b_1$ ;
      if  $l_2$  begins with "Chapter" or a number and the following several lines are not items in
      Table of Contents, List of Table, or List of Figures then
        body starts here;
      if  $l_2$  begins with "Bibliography" or "Appendix" then
        back starts here;
    end
  print buffers;

  //search for elements in buffers
  scan from the head of buffers;
  if one buffer stores "A Dissertation Presented By" or "By" then
    //the following buffer stores the author
    add 2 to wt;
    add 2 to wr;
  if find element "Report" in the buffers then
    add 2 to wt;

```

```

if find element "Thesis" in the buffers then
    add 2 to wr;
if find "Submission" then
    scan back and forth of buffers;
    if "Submission" in a block of text (there are empty lines with the last block of text and the
    next block of text in the buffers) then
        add 1 to wt;
if find "Abstract" then
    if in the following text before the other elements start, there is at least one paragraph (a
    piece of text has more than 3 lines and each line is equal to or greater than the normal line
    length(55) then
        add 2 to wt;
        add 2 to wr;
if find "Certification of Approval" then
    scan back and forth of buffers;
    if "Certification of Approval" in a block of text (there are empty lines with the last block of
    text and the next block of text in the buffers) then
        add 1 to wt;
if find "Acknowledgments" then
    if the following text has more than 3 lines and each line is equal to or greater than the
    normal line length(55) in a converted text file then
        add 1 to wt;
        add 1 to wr;
if find "Table of Contents" then
    if the following text before the other elements start is an item list and there are at least 3
    items beginning with "Chapter" or number and ending with page number then
        add 2 to wt;
        add 1 to wr;
if find "List of Tables" or "List of Figures" then
    if the following text before the other elements start is an item list and there are at least 3
    items ending with page number then
        add 1 to wt;
        add 1 to wr;
delete buffers;
//end of processing for front

//processing for body
initial a temporary buffer tb for storing chapter title, section title, or subsection title;
while (!end of dtxt and body starts and back not starts) //1
begin
    initial a new block b2 of buffers;
    add tb into a buffer of b2;
    while (!end of dtxt and next section or subsection not starts) //2
    begin
        get one line l3 from dtxt;
        put l3 into one buffer of b2;
        if l3 begins with a number and each term except for stop words and numbers begins with a
        capital letter and not a formula then
            check the following several lines ls;

```

```

    if  $l_3$  is not an item list then
         $l_3$  is a chapter title, section title, or a subsection title;
        store  $l_3$  into the temporary buffer  $tb$ ;
        next section or subsection starts here;
    if  $l_3$  begins with "Bibliography", "Appendix" or a number and the following several lines
    are not items in table of contents, list of table, or list of figures then
        back starts here;
end //while 2

print buffers;
scan each buffer  $b$  to search for chapter title  $ct$ , section title  $st$ , or subsection title  $ut$ 
if find  $ct$  then
    extract chapter number  $cn$  from  $b$ ;
    if  $cn$  is a new chapter then
        store  $ct$  into the body structure;
    if find  $st$  then
        if there is at least one paragraph under  $st$  then
            extract  $cn$  from  $b$ ;
            extract  $sn$  from  $b$ ;
            store  $sn$  into the body structure under  $ct$ ;
        if find  $ut$  then
            if there is at least one paragraph under  $st$  then
                extract  $ct$  from  $b$ ;
                extract  $st$  from  $b$ ;
                extract  $ut$  from  $b$ ;
                store  $ut$  into the body structure under  $st$  of  $ct$ ;
            delete buffers;
        end //while 1
    //end of processing for body

//processing for back
initial a block of buffers  $b_3$ ;
while (!end of  $d_{txt}$ )
begin
    get one line  $l_3$  from  $d_{txt}$ ;
    put  $l_3$  into one buffer of  $b_3$ ;
end //while
print buffers;
scan buffers of  $b_3$ ;
if find "Bibliography" then
    scan the following text  $t_1$  before appendix;
    if  $t_1$  is an item list then
        add 1 to  $wt$ ;
        add 1 to  $wr$ ;
    if find "Appendix" then
        scan the following text  $t_2$ ;
        if  $t_2$  is a piece of text then
            add 1 to  $wt$ ;
            add 1 to  $wr$ ;
        delete buffers;
    //end of processing for back

```

```

close dtxt;
check body structure;
if there is body structure then
    add 4 to wt;
    add 4 to wr;
delete body structure;

if wt>12 then
    return 1;
else if wr>10 then
    return 2;
else
    return 0;
end //thesis/report_check

```

As shown in the above algorithm, except for the body elements of DTD for theses or technical reports, for each element having #REQUIRED key word and none of its ancestors being optional, DFS adds 2 to the weight; for each element with #IMPLIED, it adds 1. For DTD of theses, the implied element “thesis” is a special case. When DFS finds the term “Thesis” on the front page, it considers the document highly probable as a thesis. Then, the weight is updated by 2. DFS add 4 to the weight of a thesis for a document if it has the body structure of thesis. This also applies to the technical reports. Therefore, the lower bound of the weight for a thesis is 12; the lower bound for a technical report is 10.

Due to the fact that there are many ways to write a given element, DFS uses semantic rules for a number of elements. The equivalent formats of some elements in DTD of thesis and technical report are as follows:

Report :- A Major Report ∨ A Major Technical Report

Contents :- Table of Contents

List of Table :- List of Tables

List of Figure :- List of Figures ∨ Table of Figures

Acknowledgments :- Acknowledgements

References :- Resources ∨ Related resources ∨ Related Publications ∨ Literature Cited

3.4.3.2 Academic Paper Structure Extraction

Most academic papers, usually published by institutes or journals, have a layout much different from theses and technical reports. In the front page of an academic paper, title is in the first block, but it often follows page number, date, author's name, and/or journal information. As mentioned in section 3.4.2.4, the abstract in an academic paper sometimes does not include the term "Abstract". It is composed in bold or italics, and/or in different font and size from the body text. However, these formats are totally lost in the converted plain text files.

Different from theses and technical reports, an academic paper has fewer layers of body structure. Sometimes the body is composed only of sections; sometimes a section also contains subsections. Compared with a thesis or a technical report, the layout of section title and subsection title in an academic paper has many variations as shown below.

- a. Only the first word begins with a capital letter
- b. All letters are in capital
- c. There is no section number nor subsection number
- d. There is space between every two letters in the section title. For example,

"1 I N T R O D U C T I O N"

The most special feature of an academic paper is its complex body layout. The body may have two or three columns. After conversion by pdftotext2.0, a paragraph in two columns is separated into two text blocks. The converter pdftotext3.0 solves the problem of dividing paragraphs. However, if a journal includes unrelated materials in the body of an

academic paper, then these materials appear as a part of the paper in the text file. Moreover, if these materials include any elements from the back part of the DTD for academic papers, these elements may be misunderstood as the back part of this paper during document structure extraction. This leads to a wrong filtering result. An example of a document with such a layout is presented in [44] in PDF and [45] in TXT format converted by pdftotext 3.0.

The back page of an academic paper consists of references and appendices, and sometimes also contains acknowledgments. Occasionally, these elements are composed as sections in the body. For instance, they are denoted as “9. Acknowledgments” and “10. References”.

To match a candidate document with the DTD of academic papers, it is separated into three parts: front, body, and back (if any), and each part is stored in a block of buffers. Next, the block is scanned back and forth to find elements of that part. Since some academic papers have no explicit element “Abstract” for their abstracts, we assume that the paragraphs after title and before the body are for the abstract. Due to the fact that some papers identify section titles according to font, font size, and font style which will be lost in the converted text file, the normal body structure extraction method of DFS based on section number cannot extract body structure for all academic papers. Therefore, we also utilize an optional method to extract body structure. In this method, DFS first scans the text file for a section title. If it finds that a line begins with a capital letter and is shorter than the next non-empty line, and the previous line is empty, the line is considered as a section title. Then, DFS scans the following text before the next title. If

there is at least one paragraph, then DFS assumes this to be a section. If DFS finds at least 3 sections, it considers that the document has the body structure of academic papers. To extract the back part such as “9 Acknowledgment” and “10 References” from the body, DFS extracts the first back element and stores it and the following text into the back buffers. The algorithm for academic paper structure extraction is shown below:

Input: a converted text document d_{txt} from the temporary repository

Output: 1 if d_{txt} is an academic paper
 0 if d_{txt} is not an academic paper for CINDI

begin

 initial body structure;
 open d_{txt} ;

 //processing for front
 pass empty lines;

 initial a block of buffers b_1 ;
 initial a temporary buffer tb for storing element in the back part or storing section title st , or subsection title ut in the body part;

 // put the front of d_{txt} into b_1

while (!end of d_{txt} and body not starts and back not starts)

begin

 get one line l_2 from d_{txt} ;

 put l_2 into one buffer of b_1 ;

if l_2 begins with a number and the first term after the number in l_2 begins with a capital letter and the following several lines are not an item list **then**

begin

 assign l_2 as l_2' after trimming the number;

if l_2' is “Bibliography” or “Appendix” or “Acknowledgments” **then**

 back starts here;

 store l_2' into the temporary buffer tb which will be put into the back block;

 //check if the following lines are references without keyword “bibliography” or “References”

else if most of the following buffers store strings beginning with '['

 back starts here;

else

 body starts here;

end

end

 print buffers;

 //search for elements in buffers

 scan from the head of buffers;

 //check title after ignore page number line, date line, and journal info line


```

if (title_check (buffers)) then
    add 2 to paper weight wp;
if find "Abstract" then
    if in the following text before the other elements start, there is at least one then
        add 2 to wp;
else
    search for text in buffers;
    if there is at least one paragraph then
        //we suppose the paragraphs before body and after title are abstract
        add 2 to wp;
if find "Keywords" then
    if there are at least 3 terms under "Keywords" then
        add 2 to wp;
if find "Table of Contents" then
    if the following text before the other element "Abstract" or the end buffer is an item list and
    each item begins with a number then
        add 1 to wp;
delete buffers;
//end of processing for front

//processing for body
while (!end of dtxt and body starts and back not starts)
begin
    initial a new block b2 of buffers;
    add tb into a buffer of b2;
    store one section or one subsection into b2; //similar with that in thesis/report_check()

    search for body structure; //similar with that in thesis/report_check() except for having no
    chapter layer
    delete buffers;
end
//end of processing for body

//processing for back
initial a block of buffers b3;
if there is an element for back in the tb then
    store the content of tb into one buffer of b3;
while (!end of dtxt and back starts)
begin
    get one line l3 from dtxt;
    put l3 into one buffer of b3;
end //while
print buffers;
scan buffers of b3;
if find "Acknowledgments" then
    scan the following text t1 before appendix;
    if there is text before the next element starts then
        add 1 to wp;
if find "Bibliography" then
    scan the following text t1 before appendix;

```

```

    if  $t_1$  is an item list then
        add 1 to wp;
    if find "Appendix" then
        scan the following text  $t_2$ ;
        if  $t_2$  is a piece of text then
            add 1 to wp;
        delete buffers;
        //end of processing for back

close  $d_{\text{txt}}$ ;
check body structure;
delete body structure;
if there is body structure then
    add 4 to wp;
else
    if optionalstructure_check() then
        add 4 to wp;

if  $wp > 6$  and there is body structure then
    return 1;
else
    return 0;
end //paper_check

```

Using the same weight assignment principle as theses/technical reports, DFS treats a document having the body structure of academic papers and the minimum weight of 6 as an academic paper. Similar to the DTDs for theses and technical reports, the DTD of academic papers also has equivalent formats for some elements as shown below.

Table of Contents :- Contents

Keywords :- keywords:

References :- Resources ∨ Related resources ∨ Related Publications ∨ Literature Cited

Acknowledgments :- Acknowledgment ∨ Acknowledgements

3.4.3.3 FAQ Structure Extraction

According to the DTD of FAQs, a FAQ includes two parts: title and contents. Sometimes title is followed by a block of text consisting of only questions. The body of a FAQ is composed of many pairs of questions and the corresponding answers. A question and its answer may be in different blocks or in one block. Each question normally ends with a

question mark. A question may be one or several lines long with possibly empty lines between them. The answer to a question is usually immediately after the question. Sometimes each pair of question and answer follows the words “Question:” and “Answer:”. After each pair of question and answer, there are usually empty lines but this is not always the case. To extract FAQ structure from a candidate document, DFS scans this document from beginning to end. When DFS finds a question, a sentence ending with “?”, it scans the following text before the next question. If the following text contains at least one paragraph, then DFS considers the question and the following text as a pair of question and answer. If DFS finds at least 6 pairs of question and answer, it treats this document as an FAQ. The algorithm of FAQ structure extraction is shown below.

Input: a converted text document d_{txt} from the temporary repository

Output: 1 if d_{txt} is a FAQ for CINDI

0 if d_{txt} is not a FAQ

begin

open d_{txt} ;

pass empty lines;

get title line lt ;

if title in several lines **then**

concatenate title lines as lt' ;

if extract “FAQ” or “Frequently Asked Questions” from lt or lt' **then**

add 2 to weight of faq wf ;

initial the count of pairs of question and answer c as zero;

while (!end of d_{txt})

begin

while (!end of d_{txt} and next question not starts)

begin

get one line l from d_{txt} ;

if l begins with “Question” **then**

add 1 to wf ;

if l ends with “?” **then**

begin

pass empty lines;

extract paragraphs;

if meet next question, a sentence ends with “?” **then**

next question start;

if there is at least 1 paragraph **then**

update c ;

```

        end //if
    end //while
end //while
if c>5 then
    return 1;
else
    return 0;
end

```

The equivalent formats of some elements for the DTD of FAQs is shown below.

FAQ :- Frequently Asked Questions

Answer :- A

Question :- Q

3.4.3.4 Overall Algorithm for DFS

When DFS starts processing a document, it makes no assumptions about its genre. It assumes that it could be one of the acceptable types of documents and tries to match it to its DTD. DFS matches the document one by one with the DTD of theses, technical reports, academic papers, and FAQs. If the document matches with one of them, DFS treats this file as an acceptable document and saves it into the permanent repository. If this document fails all of these DTDs, it is sent to the CINDI trash. For those PDF documents that cannot be converted into text files or whose converted text documents are not printable, DFS will treat them as bad files and throw them into the trash. Finally, DFS writes document quality (filter flag) in the DOWNLOAD_STATUS table. The overall algorithm of DFS is presented below.

Input: a stream of documents from the CINDI temporary repository

Output: accepted documents to the CINDI permanent repository

rejected documents to the CINDI trash

filtering flags in CINDI database to show if one document is accepted or not for CINDI

begin

create a text file f ;

create a name list f to all the files in the temporary repository;

while(!end of f)

begin

take a document d_{pdf} from f;

```

if dpdf is a PDF document then
begin
  //document processing
  check the former format dfor of dpdf;
  if dfor is MS Power Point then
    dpdf is rejected; // slide is not acceptable for CINDI
  else //1
    begin
      if cannot open dpdf then
        sign dpdf as bad document;
      else //2
        begin
          convert dpdf into text file dtxt;
          check if dpdf is printable;
          if dpdf is not printable then
            sign dpdf as bad document;
          else //3
            begin
              if (thesis/report_extract(dtxt) then
                dpdf is accepted;
              else if (paper_extract(dtxt) then
                dpdf is accepted;
              else if (faq_extract(dtxt) then
                dpdf is accepted;
              else
                dpdf is rejected;
            end //else 3
          end //else 2
        end //else 1
      // end of document processing

    //document filtering
    remove dtxt from the temporary repository;
    check the CONVERT_PDF table;
    if dpdf is in CONVERT_PDF then
      //dpdf is converted from the original document dorig by FCS
      get docID from CONVERT_PDF according to the name of dpdf;
      // to get the original name in DOWNLOAD_STATUS
      if (dpdf is accepted) then
        begin
          store dpdf into the permanent repository;
          change the mode of dpdf in permanent repository so than the following subsystem can
          access it;
          remove dpdf from temporary repository
          if dpdf is in CONVERT_PDF then
            set filter flag in DOWNLOAD_STATUS as 1(accepted document) based on docID;
            write final location into DOWNLOAD_STATUS table;
          else
            set filter flag in DOWNLOAD_STATUS as 1(accepted document);
            write final location according to dpdf name;
          end //if
        end //if
      else
        //dpdf is not in CONVERT_PDF
        //dpdf is rejected;
      end //if
    end //document filtering
  end

```

```

else // dpdf is rejected or bad
begin
  if dpdf is in CONVERT_PDF then
    put dpdf into the trash in CINDI;
    remove dpdf from the temporary repository;
    get the document location from DOWNLOAD_STATUS according to docID;
    remove dorig from the found location;
    set filter flag to 2 (rejected document) and
    set the final location to NULL according to docID;
  else
    remove dpdf from temporary repository;
    set filter flag as 2 (rejected document) and
    write final location as NULL according to the name of dpdf;
  end //else
//end of filtering
end //if
end //while
end

```

It can be seen from the above algorithm that, if the candidate document d_{pdf} is found to be one of the acceptable types of documents, then DFS treats it as an acceptable document and stores it in the permanent repository of the CINDI Robot System. At the same time, the information in the DOWNLOAD_STATUS table is updated by setting the filter flag to 1 and recording the final location for d_{pdf} as shown in Table 3.3 using the SQL statement given below.

update DOWNLOAD_STATUS set filter_flag=1,final_location="/cndoc1/pdf/" where file_name= filename

Table 3.3 Sample of Accepted Documents in the DOWNLOAD_STATUS Table

ID	file name	file type	pdf flag	filter flag	final directory
151	MUI_chapter_March8.doc	doc	1	1	/cndoc1/pdf/
152	gdb.pdf	pdf	1	1	/cndoc1/pdf/
161	webdb02.pdf	pdf	1	1	/cndoc1/pdf/
162	concordia.final.pdf	pdf	1	1	/cndoc1/pdf/
166	paper.ps	ps	1	1	/cndoc1/pdf/
170	report.txt	txt	1	1	/cndoc1/pdf/
178	extract.pdf	pdf	1	1	/cndoc1/pdf/
182	p57-quinn.pdf	pdf	1	1	/cndoc1/pdf/

On the other hand, if the document is rejected, DFS will put it into the trash in CINDI and update the information in the DOWNLOAD_STATUS table by setting the filter flag to 2 and the final location to NULL using the following SQL statement.

update DOWNLOAD_STATUS set filter_flag=2,final_location= NULL where file_name= filename

A sample of the rejected documents having filter flag 2 in the DOWNLOAD_STATUS table is shown in Table 3.4.

Table 3.4 Sample of the Rejected Documents in the DOWNLOAD_STATUS Table

ID	file name	file type	pdf flag	filter flag	final location
181	Web%20Forms_SQL.doc	doc	1	2	NULL
188	readme.txt	txt	1	2	NULL
189	test.txt	txt	1	2	NULL
191	appraisal.pdf	pdf	1	2	NULL
92	pred.txt	txt	1	2	NULL
194	week1.pdf	pdf	1	2	NULL
195	cs335tutor.pdf	pdf	1	2	NULL
198	CCSS_Budget_2003-2004.pdf	pdf	1	2	NULL

Whether the document is accepted or rejected, it will be removed from the temporary repository. If the original document is in non-PDF format, the original document will also be removed. DFS identifies if there exists an original non-pdf format for the document by checking whether the PDF document d_{pdf} has been converted by FCS. If it has, then there is an original document in CINDI repository. DFS will delete the original file from the location given in the DOWNLOAD_STATUS table.

According to the feedback of document quality set by DFS (good if the filter flag equals 1; poor if it is 2), the CINDI Robot calculates the values (gives scores) for each server or Web server directory that has valid downloaded documents and then determines the frequency of its visit to the sites. The CINDI Robot stores these information in the SITE_STATISTICS table.

3.4.3.5 Data Structures for DFS

- Block of Buffers

A block of buffers is used to store parts of a document (front, body, and back). Each block is a double linked list so that DFS can search for elements of DTDs back and forth. A block represents a node in the list which store one line of the document. A double linked list of such blocks is shown in Figure 3.10.

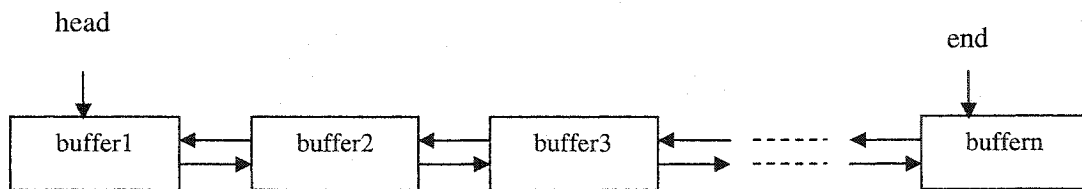


Figure 3.10 Structure of a Block of Buffers

The data structure of a buffer is shown below:

linestring, char*	next_buffer, buffer*	prev_buffer, buffer*
-------------------	----------------------	----------------------

As shown in Figure 3.10, DFS initializes a block of buffers by pointing the head pointer and the end pointer to NULL. When DFS gets one line from a document, it adds a new buffer and stores this line in this buffer. When DFS searches for elements in the block, it

scans the buffers back and forth through next_buffer and prev_buffer pointers. A sample of the block of buffers displayed by DFS is presented in Figure 3.11.

```

BUFFER:9.1Main Contributions
BUFFER:
BUFFER:Much of the research described in this thesis started out by studying what lin-
BUFFER:guists have done to address the problems encountered when building Magic and
BUFFER:PLANDoc. Using linguists' observations as a basis, techniques were developed
BUFFER:and incorporated into Casper. The following are the dissertation's major contri-
BUFFER:butions:
BUFFER:· Incorporation of a wider range of domain independent clause ag-
BUFFER:gregation operators into natural language generation systems than
BUFFER:previously possible. Many aggregation operators were analyzed and used
BUFFER:to combine propositions into one complex sentence, including quantification,
BUFFER:conjunction, adjective, prepositional phrase, and relative clause operators.
BUFFER:· Conjunction, gapping, and related ellipsis constructions are uni-
BUFFER:fied into a paradigm which is compatible with Systemic Functional
BUFFER:Grammar. Before my current conjunction algorithm was developed, it was
BUFFER:not clear how non-constituent coordinations, such as "John flew to
BUFFER:Maryland on Monday and California on Tuesday," can be incorporated into
BUFFER:Systemic Functional Grammar, one of the major formalisms used by text
BUFFER:generation community. In the above example, the sentence contains non-
BUFFER:constituent conjunction because the underlined conjuncts are not basic con-
BUFFER:stituents. By using predicate argument structure and directional constraints
BUFFER:proposed by Ross(1970) and Tai(1969), our algorithm treats simple conjunc-
BUFFER:tion, gapping, and related ellipsis constructions uniformly. As a result of
BUFFER:developing this algorithm, Casper can systematically determine which re-
BUFFER:peated constituents are redundant and delete them from the surface expres-
BUFFER:sion to make the generated sentence more concise.
BUFFER:· A corpus-based approach is used to resolve adjective ordering de-
BUFFER:cisions in sentence generation. Ordering aggregated constituents is a
BUFFER:task which must be addressed in order to produce complex but not awkward
BUFFER:sounding sentences. This is an interesting point because awkward sounding
BUFFER:sentences are grammatically correct, but humans can easily detect such dis-
BUFFER:fluencies. Generating grammatically correct sentences is not good enough; a
BUFFER:new dimension, fluency, is shown to be important in NLG and was addressed.
BUFFER:· Discourse and contextual information are utilized to select universal
BUFFER:quantifiers to make text more concise. In most NLG systems, quanti-
BUFFER:fiers are specified in the input representation. In the current work, universal
BUFFER:
BUFFER:186
BUFFER:quantifiers are derived from input representation, ontology, and discourse his-
BUFFER:tory. The proposed quantification algorithm incorporated findings from the
BUFFER:linguistic literature to ensure correct distributive reading or collective read-
BUFFER:ing is conveyed in the aggregated sentence. In addition, ambiguity related to
BUFFER:quantification operations is addressed.
BUFFER:· A corpus-based approach is used to analyze and study the sequential
BUFFER:ordering of aggregation operators. This thesis provides evidence from
BUFFER:a corpus to demonstrate the general applicability of the proposed sequential
BUFFER:ordering for aggregation operators. NLG researchers can be confident that
BUFFER:NLG systems that employ the aggregation operators in the same ordering as
BUFFER:Casper will result in grammatical, fluent, and concise sentences.
BUFFER:9.2Revisiting Issues in Clause Aggregation
BUFFER:

```

Figure 3.11 Sample of a Block of Buffers

- Body Structure

The data structure for body structure of theses, technical reports, and academic papers is a complex linked list, which consists of two levels of linked lists, level I and Level II, as shown in Figure 3.12.

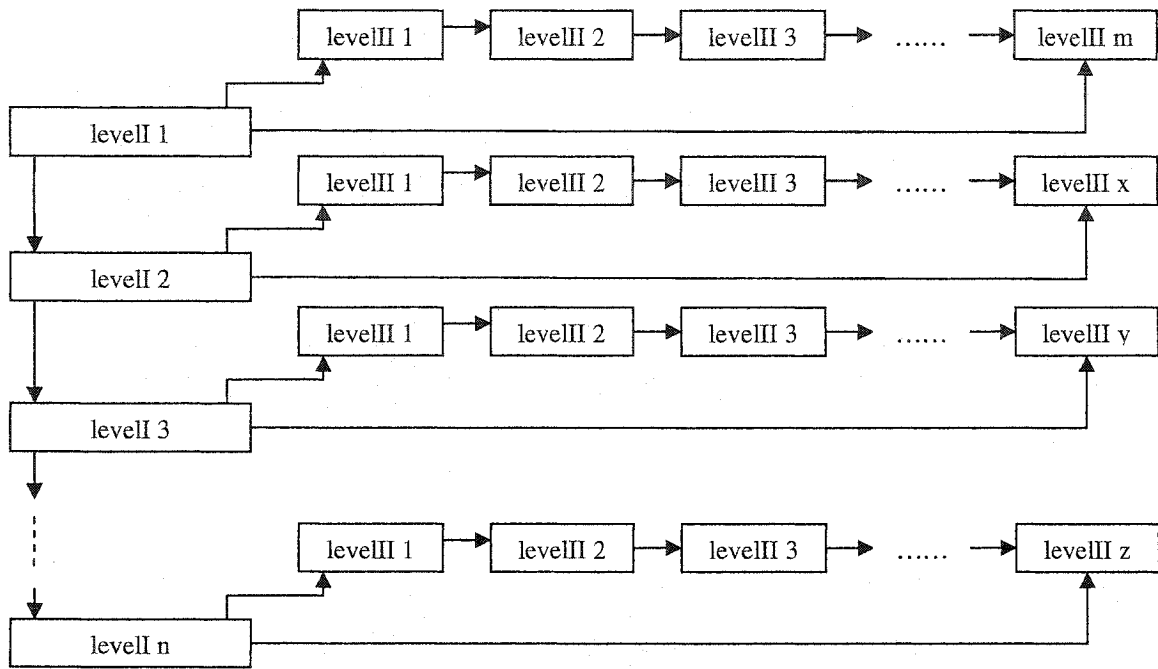


Figure 3.12 Data Structure for the Body Structure of theses, technical reports, and academic papers

As shown in Figure 3.11 , the first level list of the data structure, levelII, is for storing major components and associated pointers such as chapter information for theses, technical reports, or section information for academic papers. “levelIII” in this figure represents the second level of linked list is used to store lower level components structure such as section information of theses, technical reports, or the subsection information of academic papers. The data structure of the nodes in the first level list is presented below:

levelII_num, char*	paragraph, int	head_levelII, levelIII*	end_levelII, levelIII*	next_levelI, levelII*
--------------------	----------------	-------------------------	------------------------	-----------------------

From Figure 3.12 and the above structure, it can be seen that each node in the first level list points to a second level linked list using a head pointer (pointing to the first node in

the second level list) and an end pointer (pointing to the end of the second level list).

The data structure of the nodes in the second level linked list is shown below.

levelIII_num, char*	paragraph, int	levelIII_num, char*	next_levII, levelII*
---------------------	----------------	---------------------	----------------------

As shown in the above data structure, the data member levelIII_num is used for storing the number of subsections in a thesis, technical report, or an academic paper.

DFS stores the extracted information of body structure from a document in this complex linked list and then determine if this document has a normal body structure. A sample body structure of a thesis [46] displayed by DFS is presented in Figure 3.13.

Chapter 1: Paragraph number: 1
Chapter 2: Paragraph number: 0
Section 1: Paragraph number: 1 Subsection number:
Section 2: Paragraph number: 1 Subsection number:
Section 3: Paragraph number: 1 Subsection number: 3
Section 4: Paragraph number: 1 Subsection number:
Section 5: Paragraph number: 1 Subsection number:
Chapter 3: Paragraph number: 0
Section 1: Paragraph number: 1 Subsection number:
Section 2: Paragraph number: 1 Subsection number:
Section 3: Paragraph number: 1 Subsection number:
Chapter 4: Paragraph number: 1
Section 1: Paragraph number: 1 Subsection number: 4
Section 2: Paragraph number: 1 Subsection number: 4
Section 3: Paragraph number: 1 Subsection number: 9
Section 4: Paragraph number: 1 Subsection number: 4
Section 5: Paragraph number: 1 Subsection number: 4
Section 6: Paragraph number: 1 Subsection number: 3
Chapter 5: Paragraph number: 0
Section 1: Paragraph number: 1 Subsection number:
Section 2: Paragraph number: 1 Subsection number:
Section 3: Paragraph number: 1 Subsection number: 2
Section 4: Paragraph number: 1 Subsection number: 2
Section 5: Paragraph number: 1 Subsection number:
Chapter 6: Paragraph number: 0
Section 1: Paragraph number: 1 Subsection number:
Section 2: Paragraph number: 1 Subsection number:
Section 3: Paragraph number: 1 Subsection number: 3
Section 4: Paragraph number: 1 Subsection number: 2
Chapter 7: Paragraph number: 0
Section 1: Paragraph number: 1 Subsection number:
Section 2: Paragraph number: 1 Subsection number:
Section 3: Paragraph number: 1 Subsection number: 2

Figure 3.13 A Sample of the Body Structure for a Thesis

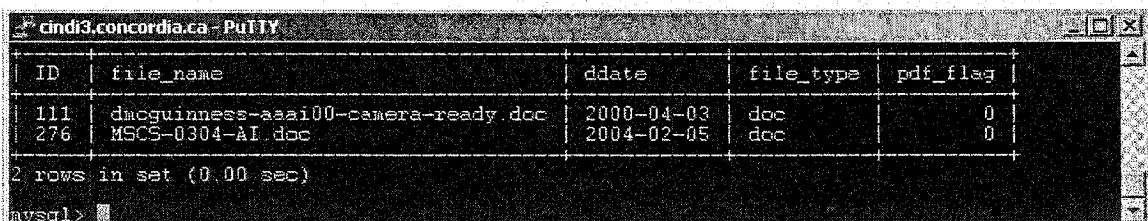
As shown in Figure 3.13, seven major components of the document under consideration (assume to be a thesis) were stored in the linked list of “levelI”; minor components of each major components were stored in the linked list of “levelII”. When searching this list, seven chapters were found and there are at least 3 sections in each chapter. Therefore, DFS considered this document has the body structure of theses based on the DTD for theses as described in section 3.4.2.1.

Chapter 4

Experiments and Evaluation

4.1 Experiment One on FCS

The FCS experiments were conducted to test its functionality. As mentioned in Chapter 3, FCS runs on a Windows platform. It is composed of three components: FCSd, Converter, and FCS Repository. The Converter consists of the Omniformat monitor and the Pdf995 PDF converter. FCSd is responsible for monitoring the CINDI Robot database and securely transferring documents; Converter is responsible for document conversion. FCSd is started after Converter runs to monitor FCS Repository. Before the experiment, as shown in Figure 4.1, the pdf_flag in the DOWNLOAD_STATUS table would be zero, while the CONVERT_PDF table would have no records for the documents to be converted.



ID	file_name	ddate	file_type	pdf_flag
111	dmoguinness-asai00-camera-ready.doc	2000-04-03	doc	0
276	MSCS-0304-AI.doc	2004-02-05	doc	0

2 rows in set (0.00 sec)

mysql>

Figure 4.1 Conversion Flag in Database before PDF Conversion

To begin the experiment, the monitor Omnifort was started to watch the FCS Repository for PDF conversions. Then, FCSd was started. The starting interface is illustrated in Figure 4.2. According to the records in the database at that time, there were 194 non-PDF documents downloaded by CINDI Robot in the non_PDF repository on the Linux platform.

```

C:\Program Files\Putty\DAEMON\Debug\DAEMON.exe
*****
*                                     *
*               FCSd of CINDI system   *
*               version 6.0             *
*               June.2004               *
*                                     *
*****

Connecting to MySQL ...      successfully!

select ID,prefix_url,file_name,temp_location,file_type,pdf_flag from DOWNLOAD_ST
ATUS where ID>0 and ID<1000 and file_type<>'tex' and file_type<>'zip' and pdf_fl
ag=0

Table select successfully..

```

Figure 4.2 The Starting Interface of FCSd

After FCSd started checking the database, it found non-PDF documents. FCSd began to transfer these documents from a Linux platform to a Windows platform, i.e. the forward transfer. The interface of the transferred documents is presented in Figure 4.3. The forward document transfer took a total of 12 minutes.

```

C:\Program Files\Putty\DAEMON\Debug\DAEMON.exe
file ID: 111
file url: 7777772E6B736C2E7374616E666F72642E6564752F70656F706C652F646C6D2F706170
6572732F646D636775696E6E6573
file name: dmcguinness-aaai00-camera-ready.doc
file location: /cndoc1/downloads/
file type: doc
PDF flag: 0

Transfer file from cindi3: dmcguinness-aaai00-camera-ready.doc
dmcguinness-aaai00-camera :      115 kB : 115.5 kB/s : ETA: 00:00:00 : 100%

file ID: 276
file url: 63732E7374616E666F72642E6564752F446567726565732F6D7363732F70726F677261
6D7368656574732F30332D30342F
file name: MSCS-0304-AI.doc
file location: /cndoc1/downloads/
file type: doc
PDF flag: 0

Transfer file from cindi3: MSCS-0304-AI.doc
MSCS-0304-AI.doc :      14 kB : 14.0 kB/s : ETA: 00:00:00 : 100%

```

Figure 4.3 Document Transfer from the Linux Platform to the Windows Platform

While FCSd executed the forward transfer jobs, Omniformat continuously monitored the FCS Repository receiving the non-PDF documents. The incoming documents in the FCS Repository were put into the converting queue as shown in Figure 4.4.

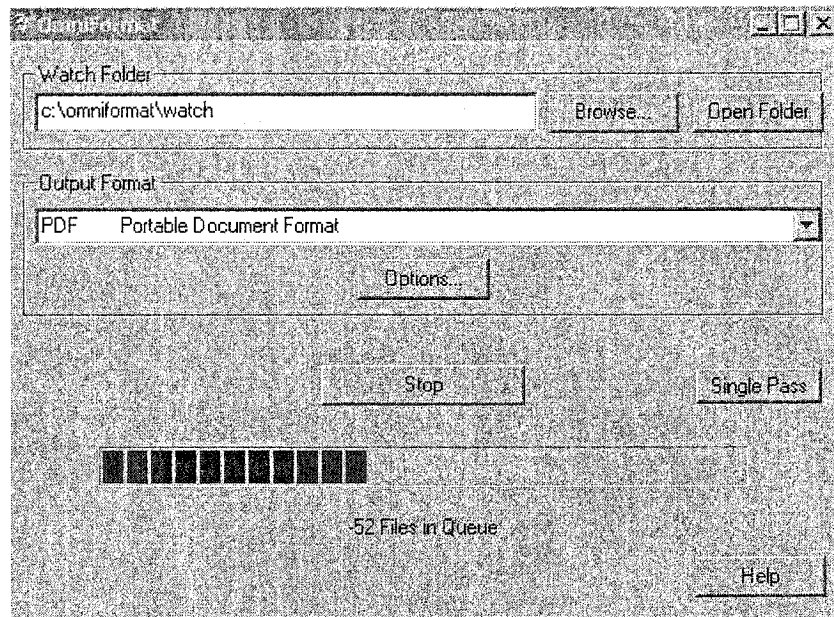


Figure 4.4 The Interface of Document Conversion

The conversion of the 194 non-PDF documents required 23 minutes.

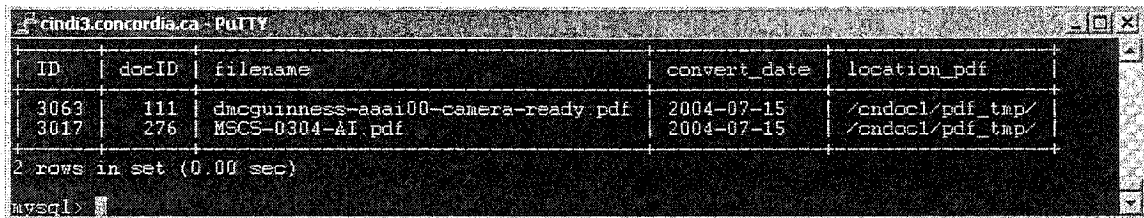
The last task of FCSd in this cycle is to transfer the converted PDF documents back to the temporary repository on the Linux platform. After one PDF document was transferred back successfully, FCSd wrote the conversion flag in the CINDI Robot database. As noted in Figure 4.5, the PDF conversion flag, pdf_flag, for the successfully converted documents was set to 1 in the database, while the converting information was recorded in the CONVERT_PDF table as shown in Figure 4.6.

ID	file_name	ddate	file_type	pdf_flag
111	dmcguinness-aaai00-camera-ready.doc	2000-04-03	doc	1
276	MSCS-0304-AI.doc	2004-02-05	doc	1

2 rows in set (0.01 sec)

mysql>

Figure 4.5 Conversion Flag after PDF Conversion



ID	docID	filename	convert_date	location_pdf
3063	111	dmagguinness-asai00-camera-ready.pdf	2004-07-15	/cndoc1/pdf_tmp/
3017	276	MSCS-0304-AI.pdf	2004-07-15	/cndoc1/pdf_tmp/

2 rows in set (0.00 sec)

mysql>

Figure 4.6 Conversion Information Recorded in the CINDI Robot Database

The total time of the transfer back to the Linux platform of the 194 documents was about 11 minutes.

After all the jobs in this cycle were finished, FCSd went to sleep for 30 minutes. Then, it woke up to check the CINDI Robot database again and the next round began. The experiment illustrated that FCS achieved the target of document conversion.

4.2 Experiment Two on DFS

Two separate tasks were conducted to test the effectiveness of filtering, that is, the ability of DFS to make the right filtering decision. One task is to test the effectiveness of the extraction of document structure through manual test. Another task is to test the performance of DFS in the real world through automatic test. All the test documents for the two tasks can be found in [47]. For a given document in either task, DFS first tries to establish the structure of the document to be either a thesis or technical report. If DFS finds this to be the case, it accepts the document for CINDI. If DFS fails to find either a thesis or a technical report structure, it attempts to compare the structure to an academic paper; if this fails, it tries FAQ structure. If the document has none of these structure, DFS would return the filtering result: value of “rejected”.

4.2.1 Manual Test

In order to test the effectiveness of the extraction of document structure, we manually produced a number of variations of scientific documents and then ran DFS, examined the output of DFS for each candidate document. Since non-readable files that do not make sense cannot be extracted by DFS, we did not try unreadable documents in the experiments. For this experiment, two theses, two technical reports, a number of academic papers, and five FAQs downloaded from the web sites were the raw documents. Some of these were in MS Word or HTML formats so that they can be used to produce a number of test cases with only some components preserved. For example, an incomplete thesis may have only the front part or the body part; a FAQ may have only questions. Since DFS recognizes PDF documents as input files, all non-pdf source documents are converted into PDF by FCS; each corresponds to one test case. The 38 test document contents are described in Table 4.1.

Table 4.1 Test Document Contents for Manual Test of DFS

4.1a Test documents with only some components of thesis or technical report DTDs

Test document	Descriptions
Document 1	having only title, author, abstract
Document 2	having only title, author, abstract, table of contents
Document 3	having only title, author, abstract, acknowledgments, and table of contents
Document 4	having only title, author, abstract, acknowledgments, table of contents, and introduction
Document 5	having only body
Document 6	having only back part (bibliography, appendix)
Document 7	having only title, author, abstract and body
Document 8	having title, author, abstract, contents, and body
Document 9	having all elements with #REQUIRED keyword and none of their ancestors are optional in the DTDs

4.1b Test documents with only some components of academic paper DTD

Test document	Descriptions
Document 1	having only title, author, abstract
Document 2	having only body
Document 3	having only back page
Document 4	having only title, author, and body
Document 5	having optional components in the body of the paper
Document 6	having back elements in the optional components in the body of the paper
Document 7	having page number before title
Document 8	having date before title
Document 9	having journal name before title
Document 10	having standard body structure (having section numbers before section titles), title, abstract, and references.
Document 11	having no space between section numbers and section titles
Document 12	having item list in body
Document 13	having labels, page numbers, and formulas in body
Document 14	initial letters in section titles being not capitals
Document 15	having no abstract, keywords, back elements, and standard body structure (having no section numbers. The difference between section titles and paragraph text is font size. All letters of section titles are capital letters).
Document 16	making back page as sections (for example, "9. Acknowledgments)
Document 17	having references but no the word "References"
Document 18	having references but key element in the last paragraph before the references in the converted text file
Document 19	having all elements include abstract but no the word "Abstract"
Document 20	having keywords but having no space between keywords and abstract
Document 21	having title, abstract, author, and references
Document 22	Layout in vertical version
Document 23	Elements are in the opposite order (back, body, and then front)
Document 24	The first page of the academic paper with total 11 pages
Document 25	having all key elements and body structure

4.1c Test documents with only some components of FAQ DTD

Test document	Descriptions
Document 1	having no key elements
Document 2	having only questions
Document 3	having some key elements of paper
Document 4	having additional information from the Web (for example, related links)
Document 5	having key elements, questions, and answers

According to the above test cases, two theses, two technical reports, one academic paper, and 5 FAQs were saved as 86 non-pdf files [47] and then converted into PDF by FCS. The test results of DFS for these test cases are presented in Table 4.2.

Table 4.2 The Results of Manual Test on DFS

4.2a The test results for theses

Test document	Test results from DFS	Correct Results (Y/N) by manual processing	Note
Document 1	Rejected	Y	
Document 2	Rejected	Y	
Document 3	Rejected	Y	
Document 4	Rejected	Y	
Document 5	Rejected	Y	
Document 6	Rejected	Y	
Document 7	Rejected	Y	This test result is difference from Document 7 in 4.2b table
Document 8	Accepted	Y	
Document 9	Accepted	Y	

4.2b The test results of technical reports

Test document	Test results from DFS	Correct Results (Y/N) by manual processing	Note
Document 1	Rejected	Y	
Document 2	Rejected	Y	
Document 3	Rejected	Y	
Document 4	Rejected	Y	
Document 5	Rejected	Y	
Document 6	Rejected	Y	
Document 7	Accepted	Y	different result from the same case of Document 7 in 4.2a
Document 8	Accepted	Y	
Document 9	Accepted	Y	

4.2c The test results of academic papers

Test document	Test results from DFS	Correct Results (Y/N) by manual processing	Note
Document 1	Rejected	Y	
Document 2	Rejected	Y	
Document 3	Rejected	Y	only references
Document 4	Accepted	Y	
Document 5	Accepted	Y	Optional components are considered as paragraphs of the body.
Document 6	Rejected	N	DFS puts all text after the first "References", which is in the optional components, into back part. Thus it cannot extract the correct body structure. Having optional structure

4.2c The Test Results of Academic Papers (Continued)

Document 7	Accepted	Y	
Document 8	Accepted	Y	
Document 9	Accepted	Y	
Document 10	Accepted	Y	
Document 11	Accepted	Y	
Document 12	Accepted	Y	A paper by Ullman, has too many lists, no formal paragraphs, most sections have only one or two sentences plus lists.
Document 13	Accepted	Y	
Document 14	Rejected	Y	The normal body structure was not extracted but it was extracted as optional.
Document 15	Rejected	N	Having optional structure but not enough elements
Document 16	Accepted	Y	
Document 17	Accepted	Y	
Document 18	Accepted	Y	The word "References" was not found but the reference list was found
Document 19	Accepted	Y	
Document 20	Accepted	Y	
Document 21	Rejected	Y	
Document 22	Rejected	Y	No element was extracted since each word is in one line in the converted text file.
Document 23	Accepted	Y	
Document 24	Rejected	Y	
Document 25	Accepted	Y	

4.2d The Test Results of FAQs

Test document	Test results from DFS	Correct Results (Y/N) by manual processing	Note
Document 1	Rejected	Y	
Document 2	Rejected	Y	
Document 3	Accepted	Y	Accepted as an academic paper. Filtering result is correct but it was misclassified
Document 4	Accepted	Y/N	Additional info such as related links and resources has no effect to extraction. Filtering is right but classification not.
Document 5	Accepted	Y	

As shown in Table 4.2, all the test results for theses and technical reports have correct filtering decision. There are abnormal results when filtering academic papers and FAQs. In test document 6 for academic paper, the document is rejected when it should be accepted because of its complex layout as shown in [44]. The document is an academic paper from IEEE journal and has optional components in the front of the body; additionally, it has references in these optional components. DFS considered the term “References” appearing alone in a line and not in a list as the beginning of the back part of an academic paper and hence treated all the text under it as the back part. Therefore, the document structure could not be correctly extracted. Another abnormal case is for test document 15. In this case, the document is rejected because its weight for paper is 2. It is possible to force it to be accepted by adjusting the paper weight rule ($wp > 6$) defined in section 3.4.3.2 to $wp \geq 2$. However, this adjustment may result in accepting wrong type of documents. For example, a proposal with title and body will be accepted as an academic paper. It is a trade-off between accepting minimum number of wrong documents and rejecting a minimum number of acceptable documents. In test document 4 for FAQs, the file was classified and accepted as an academic paper since it has key elements of academic papers.

4.2.2 Automatic Test

In this experiment, 1003 source documents were processed by DFS and produced results in one run. The source data were PDF documents directly retrieved by the CINDI Robot (729) and converted by FCS (194). To test the filtering effectiveness of DFS, the source data includes irrelevant types of documents. The test result is shown in Table 4.3.

Table 4.3 Results of Automatic Test on DFS

Total Test Documents	1003
Accepted by DFS	458
Verified to be correct	430
Verified to be wrong	28
Rejected by DFS	550
Unreadable	50
Normal	499
Verified to be correct	480
Verified to be wrong	19

In Table 4.3, “Accepted” refers to the number of documents accepted by DFS. These documents were then checked manually and taken as either correct or incorrect. Rejected documents were processed in the same way. “Correct” means the number of the filtering results that correspond to the manually checked results, while “Incorrect” refers to different results in the manually checked documents. “Unreadable” means the opened document contains mixed unrecognizable symbols that could not be read. “Normal” is the complement to “Unreadable” in “Rejected” category.

It can be seen from Table 4.3 that for accepted documents, a total of 28 incorrect filtering results were produced. The primary reason for this is that the misclassified documents were proposals (10), lecture notes (6), personal statements(5), discussion groups (4), non-technical report (2), and student instruction (1) with the main features of academic papers or FAQs. For rejected normal documents, 19 incorrect results were produced. The 19 incorrectly rejected documents are academic papers from journals. Since most incorrectly rejected documents have optional components in their body, DFS could not extract the correct body structure and they were rejected. The other documents were incorrectly rejected because of limited number of components in the documents. This experiment

shows that most theses, technical reports, and FAQs have the main structure defined in the DTDs. Consequently, they were filtered correctly. Most of the incorrectly rejected results are due to complex layout by publishers for printed papers having side bars and related articles embedded in pages of a technical paper.

To evaluate the performance of DFS, we evaluate the automatic test results, which are more similar to a real-world context. The performance is measured by how well the filtered documents match with the CINDI's expectations. For N test documents, DFS separates N into two subsets, m accepted documents and n rejected documents. Here, m is composed of correctly accepted documents (a) and incorrectly accepted documents (b); and, n consists of correctly rejected documents (c) and incorrectly rejected documents (d). So, the percentage of correct accept, S_a , is a/m ; the percentage of incorrect accept, F_a , is b/m ; the percentage of correct reject, S_r , is c/n ; the percentage of incorrect reject, F_r , is d/n . From the test results shown in Table 4.3, the filtering accuracy of DFS is calculated as shown in Table 4.4.

Table 4.4 The Filtering Accuracy of DFS from the Automatic Test

Total Test Documents, N	957(1003-46)
Accepted by DFS, m ($a+b$)	458
Correctly accepted by DFS, a	430
Incorrectly accepted by DFS, b	28
Percentage of correct accept, S_a	94% (430/458)
Percentage of incorrect accept, F_a	6% (28/458)
Rejected by DFS, n ($c+d$)	499
Correct rejected by DFS, c	480
Incorrect rejected by DFS, d	19
Percentage of correct reject, S_r	96% (480/499)
Percentage of incorrect reject, F_r	4% (19/499)

Since the 46 unreadable documents in Table 4.4 were treated as bad files and were not passed to the actual filtering processes (extracting document structure), the calculations

are based on the normal documents (957). As shown in Table 4.4, F_a is higher than the F_r . However, the primary goal of DFS is to filter out as many irrelevant documents as possible so that CINDI could return a user's query with only acceptable information. Thus, lower F_a value is more important. Hence, we tuned DFS using the following two aspects:

1. Tuned the length of a line string in paragraph checking from 50 to 55. Three lecture note and two discussion groups were filter out.
2. Tuned the weight for academic papers from 6 to 7, which means a document having title, body structure, and at least one of the other elements can be treated as an academic paper. Six proposals, four personal statements, one lecture notes, one non-technical report, one discussion groups, and one student instruction were filtered out. At the same time, three academic papers were rejected because of the presence of limited number of elements.

After tuning, there are 9 incorrectly accepted documents, consisting of 4 proposals, 2 lecture notes, 1 personal statement, 1 discussion groups, and 1 non-technical report; these have similar structure to academic papers. The filtering accuracy of DFS for the tuned parameters are calculated and shown in Table 4.5.

Table 4.5 The Filtering Accuracy after Tuning DFS

Total Test Documents, N	957(1003-46)
Accepted by DFS, m (a+b)	436
Correctly accepted by DFS, a	427
Incorrectly accepted by DFS, b	9
Percentage of success accept, S_a	98% (427/436)
Percentage of incorrect accept, F_a	2% (9/436)
Rejected by DFS, n (c+d)	521
Correct rejected by DFS , c	499
Incorrect rejected by DFS, d	22
Percentage of success reject, S_r	96% (499/521)
Percentage of incorrect reject, F_r	4% (22/521)

From Table 4.5, the percentage of failure reject documents Fa, 2%, is much lower than before (6%), which means, large number of irrelevant documents were filtered out and much less irrelevant documents were stored in the CINDI permanent repository. After tuning, DFS basically achieves its purpose.

To further assess the effectiveness of DFS, detailed statistics to the tuned DFS on the filtering accuracy of the major types of documents are given in Table 4.6. “Others” contains the irrelevant documents such as incomplete scientific papers, instructions, outlines, empty files, schedules, and name lists.

Table 4.6 Statistics on the precision of Relevant and Irrelevant Documents

	Total By manual checking	Accepted by DFS	Rejected by DFS	Accuracy (%)
Total	957			
Relevant	449			
Thesis	16	16	0	100
Technical report	27	27	0	100
Academic paper	398	376	22	94
FAQ	8	8	0	100
Irrelevant	508			
CV	18	0	18	100
E-mail	15	0	15	100
Letter	14	0	14	100
News	40	0	40	100
Form	57	0	57	100
Slide	104	0	104	100
Assignment	78	0	78	100
Discussion group	18	1	17	94
Picture	16	0	16	100
Proposal	19	4	15	79
Lecture note	55	2	53	96
Non-technical report	16	1	15	94
Others	58	1	57	98

In Table 4.6, the 957 total readable documents for the test include 449 relevant files and 508 irrelevant documents processed through manually checking. “Accepted” and

“Rejected” columns refer to the number of documents accepted by DFS and rejected by DFS respectively. The recall for each type of document is calculated by “Accepted” over “Total” of that type of document.

From the results of filtering the relevant and irrelevant documents as shown in Table 4.6, we can draw the following conclusions:

- (i) Theses, reports, and FAQs were effectively categorized in accepted documents with an accuracy of 100%. The filtering accuracy of academic papers is 94%, which means a loss of 6% papers due to the complex layout imposed by journals and less elements.
- (ii) CVs, E-mails, News, assignments, letters, forms, pictures, and slides (excluding lecture notes) were filtered out quite effectively; their accuracy is up to 100%. The primary reason affecting filtering accuracy is that the structure of the tested document is similar to the structure of academic papers. Research proposals have the lowest accuracy, 79%, since they have very similar document structure to academic papers. Some of them even have totally the same structure including title, abstract, standard body structure, and references.

Chapter 5

Conclusion and Future work

5.1 Conclusion

Due to the large volume of information on the WWW, the current global search engines cannot respond to users' queries with accurate information based on keywords or phrases. The topic specific robot is an efficient solution to this problem. It constructs a topic-specific index of Internet documents employing information-filtering technology. The goal of filtering is to maximize the number of relevant documents recommended to the repository, while minimizing the number of irrelevant ones. In the CINDI project, DFS is an application of information filtering technology based on the structure of documents. DFS selects documents of type thesis, technical report, academic paper, and FAQ, and filters out irrelevant documents such as emails, resumes, discussion groups, news, letters, and assignments. The structures for the relevant and irrelevant documents are predefined by DTDs.

DFS makes filtering decisions about the PDF documents based on the weight obtained by matching candidate documents with the DTDs of accepted types of documents. After applying the DFS on 1003 PDF documents, the final results show significant accuracy. The precision for successful reject and accept are 98% and 96% respectively. DFS basically achieved the purpose of the CINDI Robot System, to maximize the relevant documents and minimize the irrelevant documents in CINDI repository.

As a universal e-document format, PDF was chosen as the single document format in the CINDI Robot System for two reasons. One reason is to store documents in their original layout to be used by VQAS in CINDI system. Another reason is that the converter, PDFTOTEXT, is available to convert PDF documents to text files, which can be employed by DFS for extracting document structure and by VQAS for indexing. To convert all non-pdf documents downloaded by the CINDI Robot into PDF format, FCS was developed. Since CINDI was built in a Linux platform, and MS Word documents cannot be converted into PDF with high quality in the Linux platform, FCS was developed in a Windows platform. In FCS, a daemon named FCSd was implemented to monitor the CINDI Robot database and securely transfer documents between the repository in the Linux platform and the FCS Repository in the Windows platform.

5.2 Contribution of this thesis

The objective of this thesis project is to automatically filter out irrelevant documents from the retrieved documents and unify these documents' format with PDF. The design and implementation of DFS and FCS are the main contributions made by this project to the CINDI Robot System. In DFS, DTDs for theses, technical reports, scientific and technical papers, FAQs, emails, letters, resumes, and discussion groups were defined. Then, the algorithms of filtering were developed. These include overall filtering algorithm, thesis/technical report structure extraction algorithm, academic paper structure extraction algorithm, and FAQ structure extraction algorithm. FCS was designed as a converting system running on a Windows platform. In FCS, FCSd was implemented to monitor the CINDI Robot database and transfer documents between the Linux platform

and the Windows platform. To make sure FCSd automatically and securely communicates with the CINDI Robot System, FCS was configured by installing the OpenSSH server and setting MyODBC data source. To automatically convert the downloaded LaTeX documents into PDF, a script called latexconverter was implemented on the Linux platform. The design and implementation of the CINDI Robot database are composed of the efforts of several contributors. The maintenance of the converting and filtering information is another contribution of this thesis to CINDI.

5.3 Future Work

Two subsystems of CINDI Robot System developed in this thesis, DFS and FCS, basically achieved their primary purpose. However, they still need improving. DFS and FCS can be improved in several aspects.

First of all, it is necessary to improve the conversion performance of the text converter pdftotext in the future. DFS makes filtering operations based on the structure of text files. The text files are converted from PDF by the converter pdftotext. After conversion, the layouts of the converted documents are different from the original documents. For instance, sometimes there is no space between two paragraphs; two paragraphs in the same horizontal level from different columns are combined together. Also, the font type, font style, and font size are lost. These conversion problems affect the filtering effect.

Due to the fact that there exist many conventions in candidate documents corresponding to the same element in a DTD, DFS defines semantic rules for each element in DTDs. It is suggested to enrich semantic rules for each element in each type of expected documents.

Since most of academic papers come from the publications of institutes and journals, the names of publishers such as “IEEE” and “ACM” often appear on the top or the bottom of the papers. The names can be an important element for the DTD for academic papers. It is suggested to set up an academic publisher dictionary so that DFS extracts this element from documents.

Finally, FCSd in FCS was developed as a background process without users’ interaction. If users want to convert a part of the documents from the CINDI database, FCSd needs to be recompiled. To make it easy for users to choose documents, a GUI for FCSd should be developed.

References

- [1] Elizabeth Liddy, "How a Search Engine Works", *SEARCHER* Vol. 9 No. 5, May 2001.
- [2] M. Kobayashi and K. Tabkeda, "Information Retrieval on the Web", *ACM Computing Surveys*, Vol. 32, No. 2, June 2000, pp 144-173
- [3] S. Lawrence, and C. Giles, "Accessibility of information on the web", *Nature* 400, pp 107-109, 1999
- [4] B. C. Desai, S. Rajjan, "A System for Seamless Search of Distributed Information Sources", May 1994, <http://www.cs.concordia.ca/~faculty/bcdesai>
- [5] Zhan Z. "Porting the Automatic Semantic header Generator to the Web", Major report, Dept. of Computer Science, Concordia University, 2002
- [6] Yuhui W. "Enhanced Web based CINDI system", major report, Dept. of Computer Science, Concordia University, 2002
- [7] Niculae Stratica "A Natural Language Processor for Querying CINDI", master thesis, Dept. of Computer Science, Concordia University, 2002
- [8] Mohamed Amokrane Mechouet, "Web based CINDI system", master thesis, Dept. of Computer Science, Concordia University, 2001
- [9] Zhenjia Bradley Z. "CINDI Book Bag System: Design and Implementation", major report, Dept. of Computer Science, Concordia University, 2001
- [10] Xiaomer Y, "Web based CINDI system: graphical user interface design and implementation", major report, Dept. of Computer Science, Concordia University, 2001
- [11] Wen Tian, "Web-based CINDI system: database design and implementation", report, Dept. of Computer Science, Concordia University, 2001
- [12] Sami Samir Haddad, "Automatic semantic header generator", master thesis, Dept. of Computer Science, Concordia University, 1998
- [13] Nader Rajabieh Shayan, "CINDI: Concordia Indexing and Discovery System", master thesis, Dept. of Computer Science, Concordia University, 1997
- [14] Adobe PDF, Acrobat Company,
<http://www.adobe.com/products/acrobat/adobepdf.html>
- [15] Youguan Zhou, "CINDI: the virtual library graphical user interface", master thesis, Dept. of Computer Science, Concordia University, 1997

- [16] Xpdf home page, Xpdf source code and software,
<http://www.foolabs.com/xpdf/download.html>
- [17] Lawrence and C.L. Giles, "Accessibility of Information on the Web," *Nature*, vol.400, 1999, pp. 107-109
- [18] Martijn Koster, NEXOR, "Robots in the Web: threat or treat?," *ConnecXions*, Vol. 9, No. 4, April 1995. URL: <http://info.webcrawler.com/mak/projects/robots/threator-treat.html>
- [19] N.G. Shaw, A. Mian and S. B. Yadav, A comprehensive agent-based architecture for intelligent information retrieval in a distributed heterogeneous environment, *Decision Support Systems*, 32(4), pp. 401-415, 2002.
- [20] M. Wooldridge, N. R. Jennings, and D. Kinny, A methodology for agent-oriented analysis and design, In proceedings of the Third International Conference on Autonomous Agents (Agents 99), Seattle, WA, pp. 69-76, 1999
- [21] R. B. Yates and B. R. Neto, *Modern Information Retrieval*, Addison Wesley, New York, 1999.
- [22] Z. Zhang, An Agent-based hybrid framework for decision making on complex problems. Ph. D. Thesis, Deakin University, Australia, 2001.
- [23] Chun-sheng L., Cheng-qi Z., Zi-li Z., "An agent-based Intelligent System for Information Gathering from World Wide Web Environment", *IEEE Proceeding of the First International Conference on Machine Learning and Cybernetics*, Beijing November, 2002. pp1852-1857.
- [24] Y. Li, "Modern intelligent agents for Web-based information gathering", Ph. D. Thesis, Deakin University, Australia, 2000.
- [25] Niran Angkawattanawit and Arnon Rungsawang, "Learnable Crawling: An Efficient Approach to Topic-specific Web Resource Discovery",
http://pindex.ku.ac.th/file_research/Learnable_Crawler_ISCIT2002.pdf
- [26] David Evans, Clairvoyance Corporation, Pennsylvania. "The Search Engines Decade -- 1994-2003",. SearchEngine Meeting, Boston, Massachusetts, April 7-8, 2003, also in <http://www.infonortics.com/searchengines/sh03/slides/evans.pdf>
- [27] Value Filtering at Stanford, 2004 Feb 20
<http://www.diglib.stanford.edu/~testbed/doc2/ValueFiltering/valueFilterAdColor.htm>
- [28] Sergey Brin and Lawrence Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine", thesis, Stanford University, 2000,
<http://www.db.stanford.edu/pub/papers/google.pdf>

- [29] Venkat N. Gudivada, Vijay V. Raghavan, William I. Grosky, Rajesh Kananagottu, "Information Retrieval on the World Wide Web," IEEE Internet Computing, 1997.
- [30] Dan Sullivan, "Eye on the Competition-Why text mining is the key enabler of automated competitive intelligence",
<http://www.intelligententerprise.com/000908/feat1.jhtml?Requestid=7495>, Feb, 2004
- [31] Lawrence, S. and Giles, C., "Context and page analysis for improved web search". IEEE Internet Compute. 2.4, 38-46. Aug, 1998
- [32] Ian M. Soboroff, "Collaborative Filtering with LSI: Experiments with Cranfield", Technical Report TR-CS-98-01, University of Maryland, Baltimore County. November, 1998
- [33] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. GroupLens: Applying collaborative filtering to Usenet news. Communications of the ACM, 40(3):77-87, March 1997
- [34] Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating "word of mouth". In Proceedings of CHI'95-Human factors in Computing Systems, pages 210-217, Denver, CO, USA, May 1995
- [35] Byoung-Tak Z. and Yong-Woo Seo, "Personalized Web-Document Filtering Using Reinforcement Learning", CiteSeer, 2001. also in: <http://citeseer.nj.nec.com/454621.html>
- [36] Marko Balabanovic and Yoav Shoham. "Fab: Content-based, collaborative recommendation". Communications of the ACM, 40(3):66-72, March 1997.
- [37] Susan T. Dumais. "Using LSI for information filtering: TREC-3 experiments" . In Donna K. Harman, editor, Proceedings of the Third Text Retrieval Conference (TREC-3), pages 219-230, Gaithersburg, MD, November 1995. Also titled "Latent Semantic Indexing (LSI): TREC-3 Report".
- [38] Steve Lawrence, C. Lee Giles, Kurt Bollacker, "Digital Libraries and Autonomous Citation Indexing", IEEE Computer, Volume 32, Number 6, pp. 67-71, 1999.
- [39] CZ-Doc2Pdf company, <http://www.convertzone.com/doc2pdf/help.htm>
- [40] Pdfcamp company, <http://www.verypdf.com/pdfcamp/pdfcamp.htm>
- [41] Pdflib company, <http://www.pdflib.com/>
- [42] Pdf995 company, <http://www.pdf995.com/>
- [43] OpenSSH homepage, <http://www.openssh.com/>

[44] An academic paper with complex layout in PDF format,
http://www.cs.concordia.ca/~bcdesai/grads/t_zhang/doc1.pdf

[45] The converted file to [43] using pdftotext 3.0:
http://www.cs.concordia.ca/~bcdesai/grads/t_zhang/doc2.txt

[46] A Sample Thesis for Extracting Body Structure:
http://www.cs.concordia.ca/~bcdesai/grads/t_zhang/thesis.pdf

[47] Test Documents:
http://www.cs.concordia.ca/~bcdesai/grads/t_zhang/document/

Appendix A Typical Document Types

Code	Document Type
AA	AREA ACCESS REGISTER
AB	ABSTRACT
AC	ADMINISTRATIVE CLAIM
AD	ADMISSION
AE	AGREEMENT
AF	AFFIDAVIT
AG	AGENDA
AI	APPLICATION
AL	APPRAISAL
AM	AUTHORIZATION FORM
AN	ANSWER (SEE FCAP LIST)
AO	AEC/ERDA MANUAL CHAPTERS
AP	AEC STAFF PAPERS/REPORTS
AR	ARGUMENT
AS	ARCHER SUMMARIES
AT	ACTION PAPER
AU	ANNOUNCEMENT
AV	APPROVL
AX	APPENDIX, APPENDICES, OR ANNEX
BB	BIBLIOGRAPHY
BF	BRIEF
BG	BACKGROUND INFORMATION
BI	BIOGRAPHY
BK	BOOK
BL	BILLS, FEES, CHARGES
BM	BUDGET MEMO
BP	BLUE PRINTS
BR	BROCHURE
BT	BUDGET
BU	BULLETIN
CA	CHANGE ORDER
CD	COURT ORIGINATED (SEE FCAP LIST)
CE	CERTIFICATE
CF	CONFERENCES
CG	CONGRESSIONAL RECORD
CH	CONGRESSIONAL HEARING
CK	CHECK, MONEY ORDER
CL	CLAIM FOR INJURY, DAMAGE, OR DEATH
CM	COMMANDERS REPORT
CN	CONTRACT
CO	CORRESPONDENCE, LETTERS, MEMOS
CP	COMPUTER PRINTOUT
CQ	CHARTER
CR	CHART
CS	CENSUS FORMS
CT	COMPLAINT (SEE FCAP LIST)
CU	COURT TRANSCRIPTS (SEE FCAP LIST)
CV	COVER SHEET
CW	CHAPTER
CX	CROSS REFERENCE SHEET
CY	CHRONOLOGY

CZ	CATALOG
DC	DEATH CERTIFICATE/AUTOPSY
DD	DATA
DE	DECONTAMINATION REPORT
DG	DIAGRAM
DI	DISCUSSION
DK	DISK, DISKETTE WITH DATA
DL	DOSIMETRY LOG, DOSIMETRY DATA
DN	DECLARATION (SEE FCAP LIST)
DP	DEPOSITION
DR	DIRECTIVE
DT	DRAFT
DW	DRAWING/SKETCH
DY	DIARY
EA	EMPLOYEE ABSENCE REPORT
EB	EXCERPT, PARTIAL DOCUMENT
EC	EVENT CARD
ED	EXPOSURE DATA (PERSONAL)
EE	EMPLOYMENT OR UNEMPLOYMENT RECORD
EF	EXAM FORMS, MEDICAL EXAM
EH	EMPLOYMENT HISTORY FOR INDIVIDUAL 172
EI	EMPLOYEE PERSONNEL INFORMATION
EL	EQUIPMENT LIST
EM	ENVIRON MONITOR REPORT
EN	ENDORSEMENT
EO	EXECUTIVE ORDER
EP	EPIDEMIOLOGICAL STUDY
ER	ERRATA
ES	ESTIMATE SHEET, JOB ORDER
ET	EDUCATION, TRAINING
EV	EVALUATION STATEMENT
EW	EMPLOYEE WITHHOLDING RECORD
EX	EXHIBIT
EY	EMPLOYEE SERVICE RECORD
FD	FOLDER, FOLDER COVER
FI	FEDERAL INSPECTION REPORT
FL	FIELD LOG BOOK, RAD LOGS
FM	FORM
FO	FIELD ORDER
FP	FRC STAFF PAPERS/REPORTS
FS	FACT SHEET
GJ	'GREAT JOB' LETTER
GL	GOVERNMENT LAW/REGULATION
GR	GRAPHS, FIGURES, VIEW GRAPHS
HB	HANDBOOK
HH	HEARING
HN	HANDWRITTEN NOTE
IC	INVOICE
IE	INCIDENT REPORT (RADIATION)
IJ	ACCIDENT/INJURY CLAIM, REPORT
IN	INSTRUCTIONS
IR	INSPECTION REPORT
IT	INTERROGATORIES
IV	INTERVIEW
IX	INDEX, INDICES
JA	JOURNAL ARTICLE
JD	JOB DESCRIPTION

LA	LABEL
LB	LOGS OR LOGBOOKS
LD	LEGAL DECISION
LE	LEGAL (GENERAL)
LG	LEGISLATION, CONGRESSIONAL BILL
LI	LISTING
LL	LISTING
LR	LABORATORY SERVICES REQUEST
LS	LEASE AGREEMENT
LT	LABORATORY REPORT, LAB DATA
MA	MAGAZINE ARTICLE
MC	MINE LEASE OR CONTRACT
MD	MINE CARDS
ME	MEDICAL LOG
MF	MINING FIELD REPORT
MI	MAIN RECORD
MJ	MEMORANDUM (LEGAL) (SEE FCAP LIST)
ML	MONITORING LOG
MM	MEETING MINUTES
MN	MANUAL
MO	MILITARY ORDER 115
MP	MAP
MR	MEDICAL RECORDS
MS	MEDICAL SERVICES REQUEST
MT	MOTION (SEE FCAP LIST)
MU	MEMORANDUM OF UNDERSTANDING
NA	NEWSPAPER ARTICLE
NB	NOTEBOOK
NC	CONFIRMATION NOTICE
ND	NOTICE OF DEPOSITION (FCAP LIST)
NG	NOTICE (GENERAL)
NL	NEWSLETTER
NM	NOTICE LETTER (PHS TO MINERS)
NO	NOTIFICATION
NP	NOTICE LETTER (PHS TO DR./HOSPITAL)
NS	NOTES
NT	NEWS TRANSMITTAL (NEWSTAB)
OB	OBITUARY
OC	ORGANIZATIONAL CHART, ORGANIZATION
OH	OCCUPATIONAL HISTORIES
OM	OPERATING MANUAL
OO	OPERATION ORDER
OP	OPPOSITION
OR	COURT ORDER
OT	OUTLINE
OV	OVERFLOW RECORD
PA	PATHOLOGY REPORTS
PB	PUBLICATION
PC	PICTURE (PHOTOGRAPH, MOVIE, ETC.)
PD	PUBLISHED DOCUMENT
PE	PROCEDURE
PF	PATENT
PG	PLEADINGS (SEE FCAP LIST)
PH	CERTIFICATE FOR PATENT CLEARANCE
PI	PRETRIAL ORDER (SEE FCAP LIST)
PL	PERSONNEL LISTING
PM	PERSONAL NOTE

PN	PLAN
PO	OPS PLAN(S)
PP	PROPOSAL
PQ	PRODUCTION REQUEST
PR	PRESS RELEASE, PUBLIC ANNOUNCEMENT
PS	PLAINTIFF MINE SUMMARIES
PT	PERMITS
PU	PURCHASE ORDER
PX	PRESENTATION, PROGRAM, LECTURE
PY	POLICY STATEMENT
QA	QUALITY ASSURANCE
QR	PROCEDURE QUALIFICATION RECORD
QS	QUESTIONS/ANSWERS
QU	QUESTIONNAIRE
RA	REVIEW ACTION SHEET
RC	RECOMMENDATION
RD	RAD SURVEY
RE	RESOLUTION
RF	RADON SAMPLE FORM
RG	REGULATIONS
RH	RADIATION EXPOSURE HISTORY
PJ	PAPER, UNPUBLISHED RESEARCH
PK	PROTOCOL
RL	RADIO LOG
RN	REQUEST FOR ADMISSION (FCAP LIST) (58)
RO	ROUTING SLIP
RP	RECEIPT
RQ	REQUEST
RR	REAL ESTATE RELATED
RS	RESUME
RT	REPORT
RW	RELEASE FOR/FROM WORK, ASSIGNMENT
SA	STATEMENT
SC	SCHEDULE, CALENDER
SE	SPEECH
SI	STATE INSPECTION REPORT
SL	SPECIAL INSTRUCTIONS
SM	SUMMONS
SN	SPECIFICATION
SO	STANDARD OPERATING PROCEDURES
SP	SUBPOENA
SQ	STIPULATION
SR	SCIENTIFIC DIRECTORS REPORT
SS	SOCIAL SECURITY SUMMARIES
ST	STRIPCHART
SU	SUMMARY
SV	SURVEY
SW	SWORN STATEMENT (SEE FCAP LIST)
TB	TABLE
TC	TIME CARD
TE	TEST BULLETIN
TF	TAX FORM
TG	TELEGRAM
TH	TELEPHONE CONVERSATION NOTES
TI	TECHNICAL INSTRUCTIONS
TL	TRIAL
TM	TEST MANAGERS REPORT

TN	TERMINATION
TP	TRIP REPORT
TR	TRANSCRIPT SPEECH, COURT STATEMENT
TS	TRANSMITTALS
TT	TABLE OF CONTENTS
TY	TESTIMONY
UC	UNEMPLOYMENT CLAIM
VO	VERBAL ORDER
VT	VIDEOTAPE
WC	WORKER'S COMPENSATION CLAIM
WL	WORKING LEVELS ESTIMATE
WO	WORK ORDER
WT	WITNESS RECORD
XY	X-RAY, CAT SCAN
ZZ	MISCELLANEOUS

Note: this table is from: <http://worf.eh.doe.gov/> .

Appendix B Source Formats Accepted by Omniformat and Pdf995

Format	Description	Notes	Export
ANS	Text with Layout	<i>Available if Word2000 is installed*</i>	
ART	PFS: 1st Publisher	Format originally used on the Macintosh and later used for PFS: 1st Publisher clip art.	
ASC	Text with Layout	<i>Available if Word2000 is installed*</i>	
AVI	Microsoft Audio/Visual Interleaved		
AVS	AVS X image		
BMP	Microsoft Windows bitmap		✓
CMYK	Raw cyan, magenta, yellow, and black samples		✓
CUT	DR Halo		
DCM	Digital Imaging and Communications in Medicine (DICOM) image	Used by the medical community for images like X-rays.	
DCX	ZSoft IBM PC multi-page Paintbrush image		
DIB	Microsoft Windows Device Independent Bitmap	DIB is a BMP file without the BMP header. Used to support embedded images in compound formats like WMF.	✓
DOC	Word Document	<i>Available if Word2000 is installed*</i>	
DPX	Digital Moving Picture Exchange		
EMF	Microsoft Enhanced Metafile (32-bit)		
EPDF	Encapsulated Portable Document Format		✓
EPI	Adobe Encapsulated PostScript Interchange format		✓
EPS	Adobe Encapsulated PostScript		✓
EPSF	Adobe Encapsulated PostScript		✓
EPSI	Adobe Encapsulated PostScript Interchange format		
EPT	Adobe Encapsulated PostScript Interchange		✓

	format with TIFF preview		
FAX	Group 3 TIFF	See TIFF format. Note that FAX machines use non-square pixels which are 1.5 times wider than they are tall but computer displays use square pixels so FAX images may appear to be narrow unless they are explicitly resized using a resize specification of "150x100%".	✓
FITS	Flexible Image Transport System		
FPX	FlashPix Format		✓
G3	Group 3 FAX		✓
GIF	CompuServe Graphics Interchange Format	8-bit RGB PseudoColor with up to 256 palette entries.	✓
GRAY	Raw gray samples		
HPGL	HP-GL plotter language		
HTML	Hyper-Text Markup Language	<i>Available if Word2000 is installed*</i>	
ICO	Microsoft icon	Also known as "ICON".	
JBIG	Joint Bi-level Image experts Group file interchange format		✓
JNG	Multiple-image Network Graphics	JPEG in a PNG-style wrapper with transparency.	
JP2	JPEG-2000 JP2 File Format Syntax		✓
JPC	JPEG-2000 Code Stream Syntax		✓
JPEG/JPG	Joint Photographic Experts Group JFIF format		✓
MAT	MATLAB image format		
MHTML	Web Archive	<i>Available if Word2000 is installed*</i>	
MONO	Bi-level bitmap in least-significant-byte first order		
MNG	JPEG Network Graphics		
MPEG/MPG	Motion Picture Experts Group file interchange format (version 1)		✓
M2V	Motion Picture Experts Group file interchange format (version 2)		✓

MTV	MTV Raytracing image format		
OTB	On-the-air Bitmap		
P7	Xv's Visual Schnauzer thumbnail format		
PAL	16bit/pixel interleaved YUV		✓
PALM	Palm pixmap		✓
PBM	Portable bitmap format (black and white)		✓
PCD	Photo CD	The maximum resolution written is 768x512 pixels since larger images require huffman compression (which is not supported).	✓
PCDS	Photo CD	Decode with the sRGB color tables.	
PCL	HP Page Control Language	For output to HP laser printers. Write only.	✓
PCT	Apple Macintosh QuickDraw/PICT		✓
PCX	ZSoft IBM PC Paintbrush file		
PDB	Palm Database ImageViewer Format		
PDF	Portable Document Format		✓
PFA	Postscript Type 1 font (ASCII)	Opening as file returns a preview image.	
PFB	Postscript Type 1 font (binary)	Opening as file returns a preview image.	
PGM	Portable graymap format (gray scale)		
PICON	Personal Icon		
PICT	Apple Macintosh QuickDraw/PICT file		
PIX	Alias/Wavefront RLE image format		
PNG	Portable Network Graphics		✓
PPM	Portable pixmap format (color)		
PPT	PowerPoint Presentation	<i>Available if PowerPoint 2000 or later is installed.</i>	
PS	Adobe PostScript file		✓

PS2	Adobe Level II PostScript file		✓
PS3	Adobe Level III PostScript file		✓
PSD	Adobe Photoshop bitmap file		✓
PTIF	Pyramid encoded TIFF	Multi-resolution TIFF containing successively smaller versions of the image down to the size of an icon. The desired sub-image size may be specified when reading via the -size option.	
PWP	Seattle File Works multi-image file		
RGB	Raw red, green, and blue samples (8 or 16 bits, depending on the image depth)		✓
RLA	Alias/Wavefront image file		
RLE	Utah Run length encoded image file		
RTF	Rich Text Format	<i>Available if Word2000 is installed*</i>	
SCT	Scitex Continuous Tone Picture		
SFW	Seattle File Works image		
SGI	Irix RGB image		
SUN	SUN Rasterfile		
SVG	Scalable Vector Graphics		✓
TGA	Truevision Targa image	Also known as formats "ICB", "VDA", and "VST".	
TIFF	Tagged Image File Format	Also known as "TIF".	✓
TIM	PSX TIM file		
TXT	Raw text file		
VICAR	VICAR rasterfile format		
VIFF	Khoros Visualization Image File Format		
WBMP	Wireless bitmap	Support for uncompressed monochrome only.	✓
WK4	Lotus 1-2-3	<i>Available if Word2000 is installed. Requires Word Import Converter*</i>	
WPD	Word Perfect	<i>Available if Word2000 is installed. Requires Word Import Converter*</i>	
WPG	Word Perfect Graphics File		

WRI	Microsoft Write	<i>Available if Word2000 is installed*</i>	
XBM	X Windows system bitmap, black and white only	Used by the X Windows System to store monochrome icons.	
XCF	GIMP image		
XLS	Excel	<i>Available if Excel2000 or later is installed.</i>	
XML	Extensible Markup Language		
XPM	X Windows system pixmap	Also known as "PM". Used by the X Windows System to store color icons.	
XWD	X Windows system window dump	Used by the X Windows System to save/display screen dumps.	

1. A paragraph with table

Minimization: Note that u_1 and u_2 cannot be eliminated because every homomorphism is the identity on u and therefore on x_1 and x_6 . However, the other 2 rows can be eliminated using the homomorphism that maps x_8 to x_2 and x_9 to x_3 , and is the identity everywhere else. Thus the minimal tableau looks as following:

The equivalent SPJR query with minimal number of joins: $\Pi_{AC}(\Pi_{AB}(R) \bowtie \Pi_{BC}(R))$

3. Some Special symbols:

$(\pi_A(R))$	ϕ	ψ	ζ	δ	σ	ω	$free(\psi)$	\forall	X_d
Π	δ	ρ	\times	$\delta_{Director \rightarrow D}$	(rename)	\bowtie	\leftarrow	\downarrow	
\wedge	\models	\subseteq	\exists	\in	\cup	\vee	\neq	\leq	\geq
\cap	\in	\pm	$+$	$-$	\times	\div	\therefore	\therefore	

4. A paragraph with hyperlink

Dr. Bipin C. DESAI: Web Publications

- **Revisited**
- Indexing and Searching Virtual Libraries: CIC-Forum on America in the Age of Information
- Econsumer
- Econsumer Talk Slides
- ASHG: Automatic Semantic Header Generator: Report
- Think it over: Author Anonymous!

5. An image with JPG format



1. A paragraph with table

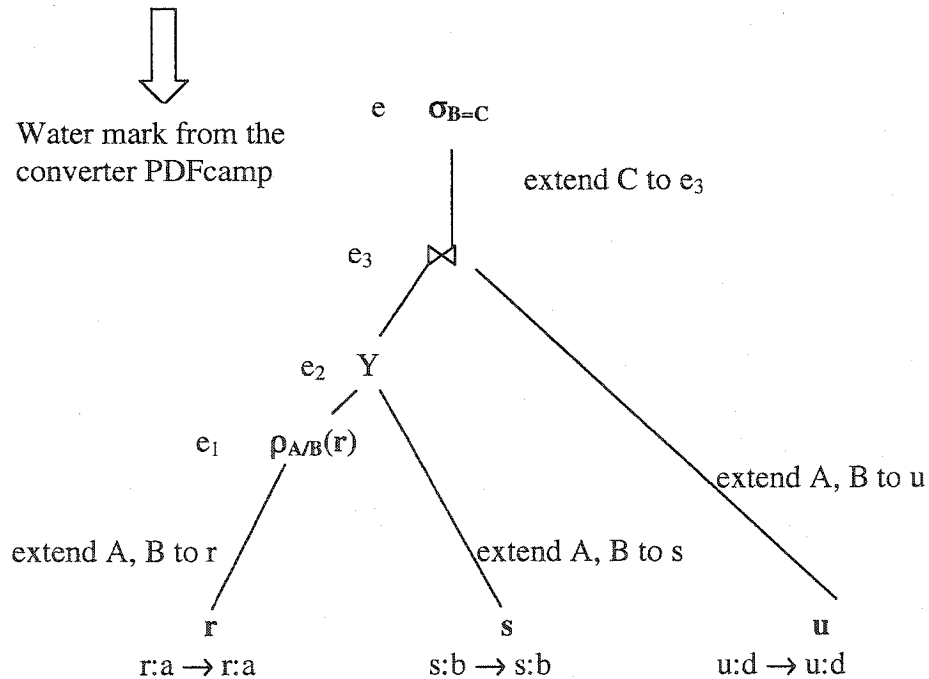
A	B	C	D	E
x ₁	x ₃	x ₂	x ₆	x ₄
x ₃	x ₂	x ₁	x ₄	x ₆
x ₂	x ₃	x ₆	x ₄	x ₁

Minimization: Note that u_1 and u_2 cannot be eliminated because every homomorphism is the identity on u and therefore on x_1 and x_6 . However, the other 2 rows can be eliminated using the homomorphism that maps x_8 to x_2 and x_9 to x_3 , and is the identity everywhere else. Thus the minimal tableau looks as following:

A	B	C
x ₁	x ₂	x ₃
x ₄	x ₂	x ₆
x ₁		x ₆

The equivalent SPJR query with minimal number of joins: $\Pi_{AC}(\Pi_{AB}(R) \bowtie \Pi_{BC}(R))$

2. Figure



3. Some Special symbols:

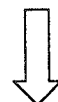
$(\pi_A(R))$	φ	ψ	ζ	δ	σ	ω	$free(\psi)$	\forall	X_d
Π	δ	ρ	\times	$\delta_{Director \rightarrow D}$	$(rename)$	\bowtie	\leftarrow	\downarrow	
\wedge	\models	\subseteq	\exists	\in	\cup	\vee	\neq	\leq	\geq
\cap	\in	\pm	$+$	$-$	\times	\div	\therefore	\therefore	

4. A paragraph with hyperlink

Dr. Bipin C. DESAI: Web Publications

- [Revisited](#)
- [Indexing and Searching Virtual Libraries: CIC-Forum on America in the Age of Information](#)
- [Econsumer](#)
- [Econsumer Talk Slides](#)
- [ASHG: Automatic Semantic Header Generator: Report](#)
- [Think it over: Author Anonymous!](#)

5. An image with JPG format



Water mark from the
converter PDFcamp

1. A paragraph with table

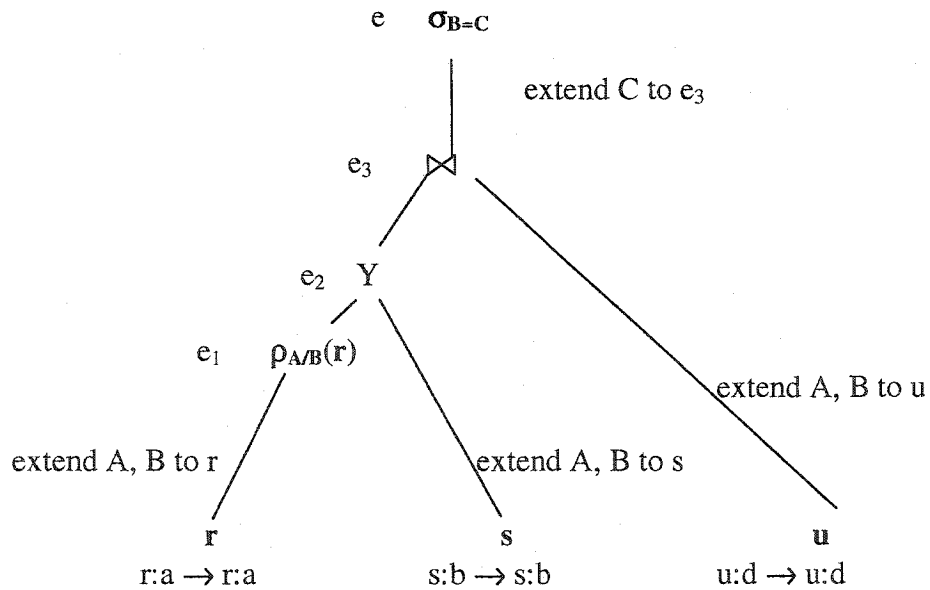
A	B	C	D	E
x ₁	x ₃	x ₂	x ₆	x ₄
x ₃	x ₂	x ₁	x ₄	x ₆
x ₂	x ₃	x ₆	x ₄	x ₁

Minimization: Note that u_1 and u_2 cannot be eliminated because every homomorphism is the identity on u and therefore on x_1 and x_6 . However, the other 2 rows can be eliminated using the homomorphism that maps x_8 to x_2 and x_9 to x_3 , and is the identity everywhere else. Thus the minimal tableau looks as following:

A	B	C
x ₁	x ₂	x ₃
x ₄	x ₂	x ₆
x ₁		x ₆

The equivalent SPJR query with minimal number of joins: $\Pi_{AC}(\Pi_{AB}(R) \bowtie \Pi_{BC}(R))$

2. Figure



3. Some Special symbols:

$(\pi_A(R))$	ϕ	ψ	ζ	δ	σ	ω	$free(\psi)$	\forall	X_d
Π	δ	ρ	\times	$\delta_{Director \rightarrow D}$	$(rename)$	\bowtie	\leftarrow	\downarrow	
\wedge	\models	\subseteq	\exists	\in	\cup	\vee	\neq	\leq	\geq
\cap	\in	\pm	$+$	$-$	\times	\div	\therefore	\therefore	

4. A paragraph with hyperlink

Dr. Bipin C. DESAI: Web Publications

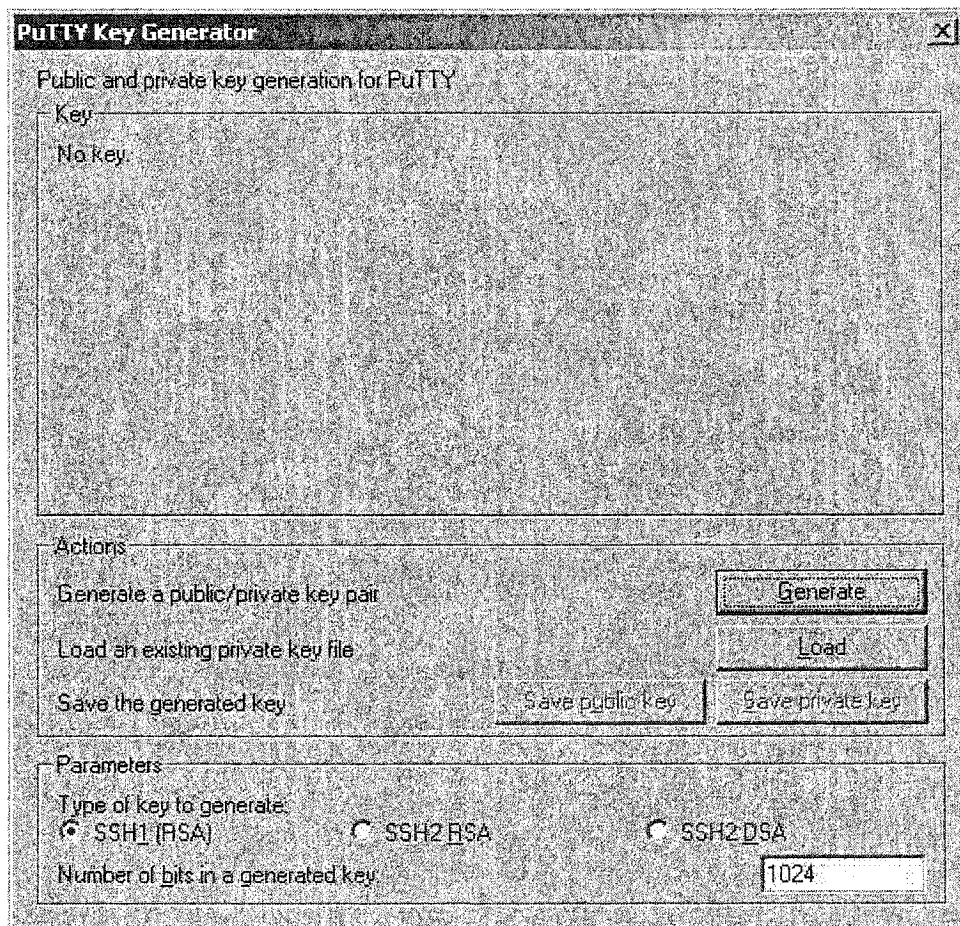
- [Revisited](#)
- [Indexing and Searching Virtual Libraries: CIC-Forum on America in the Age of Information](#)
- [Econsumer](#)
- [Econsumer Talk Slides](#)
- [ASHG: Automatic Semantic Header Generator: Report](#)
- [Think it over: Author Anonymous!](#)

6. An image with JPG format



Appendix F Key Generation for the Communication between Windows and Linux

1. Get *pageant.exe* and *puttygen.exe* from [1] and place them in *C:\Program Files\Putty*
2. In a DOS shell, enter *C:\Program Files\Putty*, run *puttygen*, and finish key generation according the following steps:
 - 1). Press the **Generate** button and follow instructions.



- 2). Type in a pass word and press the **Save private key** button and provide the file name *identity*.
 - 3). Press the **Save public key** button and provide file name *identity.pub*
3. Send *identity.pub* to Linux (*.ssh/*) and change the name as *authorized.keys*

[1] <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>