

**AUTOMATIC CLASSIFICATION OF DOCUMENTS WITH AN IN-DEPTH
ANALYSIS OF INFORMATION EXTRACTION AND AUTOMATIC
SUMMARIZATION**

by

Joseph Brandon Hohm

Bachelor of Science in Management Science
Massachusetts Institute of Technology

Submitted to the Department of Civil and Environmental Engineering in Partial
Fulfillment of the Requirements for the Degree of

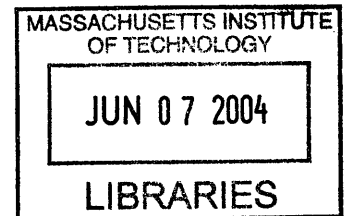
MASTER OF ENGINEERING IN CIVIL AND ENVIRONMENTAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

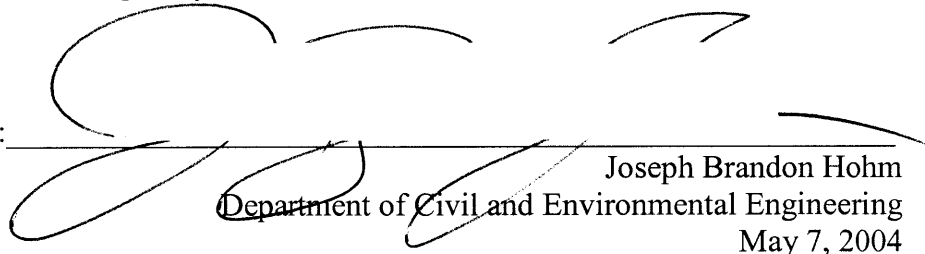
June 2004

© 2004 Joseph Brandon Hohm. All rights reserved.

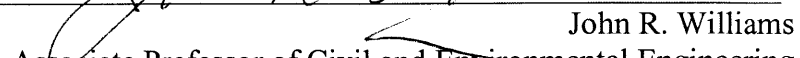


*The author hereby grants MIT permission to reproduce and to distribute publicly paper
and electronic copies of this thesis document in whole or in part.*

Signature of Author: _____


Joseph Brandon Hohm
Department of Civil and Environmental Engineering
May 7, 2004

Certified by: _____


John R. Williams
Associate Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by: _____


Heidi Nepf
Chairman, Departmental Committee on Graduate Studies

BARKER

**AUTOMATIC CLASSIFICATION OF DOCUMENTS WITH AN IN-DEPTH
ANALYSIS OF INFORMATION EXTRACTION AND AUTOMATIC
SUMMARIZATION**

by

Joseph Brandon Hohm

Submitted to the Department of Civil and Environmental Engineering on May 7, 2004 in
Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Civil and Environmental Engineering

ABSTRACT

Today, annual information fabrication per capita exceeds two hundred and fifty megabytes. As the amount of data increases, classification and retrieval methods become more necessary to find relevant information. This thesis describes a .Net application (named I-Document) that establishes an automatic classification scheme in a peer-to-peer environment that allows free sharing of academic, business, and personal documents. A Web service architecture for metadata extraction, Information Extraction, Information Retrieval, and text summarization is depicted. Specific details regarding the coding process, competition, business model, and technology employed in the project are also discussed.

Thesis Supervisor: Dr. John R. Williams

Title: Associate Professor of Civil and Environmental Engineering

Acknowledgements

I would like to dedicate this thesis to my parents Joseph and Terry Hohm for their unwavering love and support, and for serving as incredible role models throughout my life.

I would like to thank my project group, Mohan K. Akula, Vishal S. Saxena, and Sapna D. Tyagi, for their efforts, cooperation, and hard work during this entire school year.

Special thanks go to Dr. John Williams for giving me advice with regards to work, this project, my thesis, and general life problems, and being a great friend, teacher, and mentor.

I would also like to Dr. George A. Kocur for his guidance on my schedule and class work this year.

Finally, I would like to thank my friends and remaining family for their support and encouragement. Thank you for being there for me when I needed you the most, and providing consistent distractions to sustain my positive spirits.

Table of Contents

Abstract	2
Acknowledgements	3
Table of Contents	4
List of Figures	7
List of Tables	8
1. Introduction	9
1.1 Problem Statement.....	9
1.2 Research Objective and Project Overview.....	9
1.3 Thesis Organization.....	12
2. Code and Program Research	14
2.1 Code Estimation.....	14
2.1.1 Function Points.....	14
2.1.1.1 Web Page Estimation.....	15
2.1.1.2 Application Estimation.....	16
2.1.2 Lines of Code.....	17
2.1.3 Timeline.....	17
2.1.4 Code Estimation Conclusions and Validity of Results.....	18
2.2 UML Design.....	19
3. System Research	26
3.1 Classification System Design.....	26
3.2 Information Retrieval Overview.....	27
3.3 Information Extraction Overview.....	31

3.4 The I-Document System.....	32
4. Text Summarization	35
4.1 Text Summarization Overview.....	35
4.2 Summarization Research.....	35
4.3 I-Document’s Text Summarization.....	37
4.4 Improvement of the System.....	39
4.5 Summarization Conclusion.....	43
5. Metadata.....	44
5.1 Metadata Introduction.....	44
5.2 Metadata Decision and Implementation.....	44
5.3 Metadata Extension.....	47
6. Technical Overview of I-Document.....	48
6.1 I-Document.....	48
6.2 Underlying Technology.....	48
6.2.1 .NET and ASP.NET.....	48
6.2.2 Peer-To-Peer Computing.....	49
6.2.3 Web Services.....	51
6.2.4 Microsoft Office.....	52
6.3 Pieces of I-Document.....	53
6.3.1 I-Document Web Services.....	53
6.3.2 Web Application.....	56
6.3.3 Database.....	59
6.4 Assessment of the System.....	61

6.4.1 Information Retrieval Assessment.....	61
6.4.2 Information Extraction Assessment.....	63
7. Business Competition.....	64
7.1 I-Document Competition.....	64
7.2 Internet Search Engines.....	64
7.3 Data Repositories/Digital Libraries.....	66
7.4 Small Commercial Systems.....	68
7.5 Other P2P Applications.....	69
8. Business Competition Models.....	71
8.1 Attacking the Market.....	71
8.2 Customer Segmentation.....	71
8.3 Competitive Advantages.....	72
8.4 Business Forces.....	74
8.5 Competition Conclusion.....	76
9. Conclusion.....	77
9.1 Thesis Conclusion.....	77
10. References.....	78
11. Appendices.....	81
11.1 Appendix 1.....	81
11.2 Appendix 2.....	90

List of Figures

Figure 1.2- 1: Grid/P2P environment of I-Document.....	10
Figure 1.2- 2: Classification of Local Documents.....	11
Figure 2.2- 1: Web Site Activity Diagram.....	22
Figure 2.2- 2: Application Activity Diagram.....	23
Figure 2.2- 3: Client/Server Activity Diagram.....	24
Figure 2.2- 4: P2P Activity Diagram.....	25
Figure 5.2- 1: I-Document Metadata Tab.....	46
Figure 6.2.2- 1: I-Document's P2P Computing Environment.....	50
Figure 6.3.1- 1: Central Web Service.....	54
Figure 6.3.1- 2: Local Web Service.....	56
Figure 6.3.2- 1: Main Web Application Picture.....	57
Figure 6.3.2- 2: I-Document Search Page.....	58
Figure 6.3.3- 1: I-Document Database.....	60
Figure 8.3- 1: I-Document Competitive Advantage Chart.....	73
Figure 8.4- 1: 5-Forces Model for I-Document.....	76

List of Tables

Table 2.1.1.1- 1: Function point definition for the I-Document Portal.....	15
Table 2.1.1.1- 2: Function point estimation for the I-Document Portal.....	15
Table 2.1.1.2- 1: Function point definition for the I-Document Application.....	16
Table 2.1.1.2- 2: Function point estimation for the I-Document Application.....	16
Table 2.1.2- 1: Corresponding lines of code per function point.....	17

1.1 Problem Statement

Information access is increasingly becoming a problem with the exponential growth of digital information. A number of solutions have been proposed, yet there is no forefront leader. While methods are being refined and optimized, knowledge seekers still encounter problems with current search and document recovery environments.

There are several solutions and theories in the fields of string processing, text mining, and data storage. There exist Internet search engines that perform full-text searches such as Google, Yahoo!, and AlltheWeb. The engines, however, take at least a fortnight to update registries and impose inefficient page ranking algorithms to identify valued information. Conventional data repositories are popular among large firms, libraries, and educational institutions, but they come with maintenance and reliability issues. Small commercial solutions carry great costs and focus on a private rather than a public market. Finally, peer-to-peer (P2P) environments concentrate on sharing popular media like music and movies rather than on typed information. While these are solid, justified solutions, each has significant setbacks. This project recognizes these ongoing problems and seeks a different type of Information Retrieval System.

1.2 Research Objective and Project Overview

The main objective of this research is to investigate current Information Extraction and Retrieval techniques, evaluate their benefits and drawbacks, and code an application in

Web services to solve some existing retrieval problems. The result is an application called I-Document which utilizes a .Net platform coupled with a P2P sharing environment. This program automatically classifies documents on individual machines, shares them among a grid of users, and searches and retrieves other users' documents.

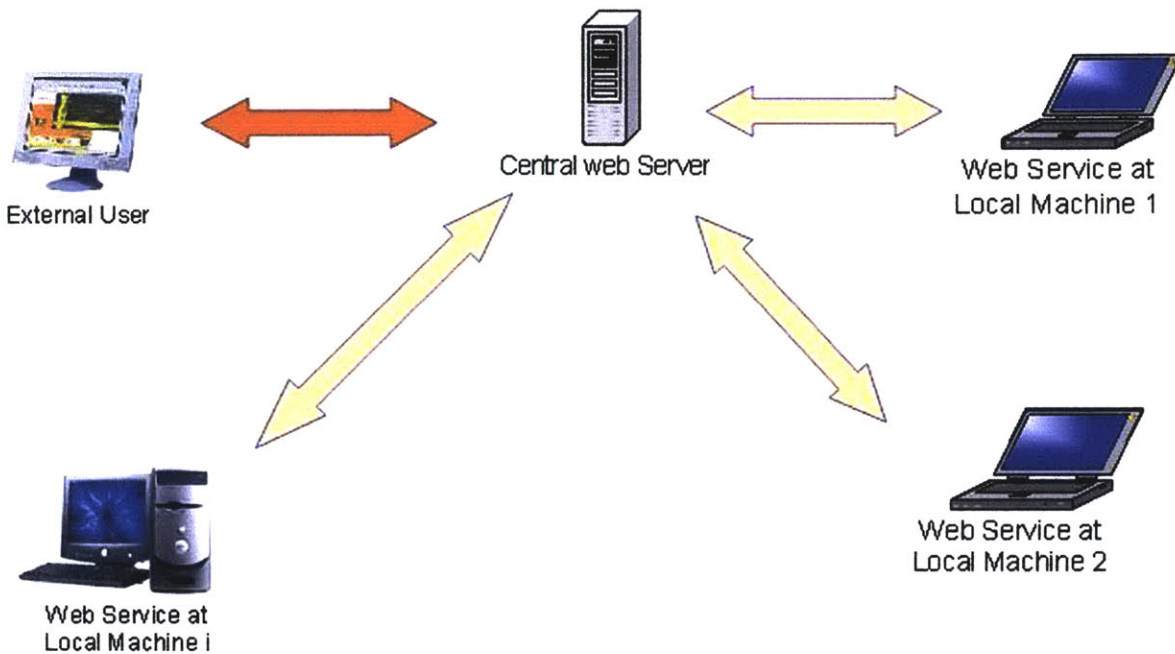


Figure 1.2- 1: Grid/P2P environment of I-Document

Utilizing the I-Document service, users can log on to the sharing network, begin the automatic classification, and freely download other documents on the grid. Upon signing in, each user's computer classifies the files in his or her shared folder according to the imposed summarization scheme. The application then submits the highlights of this classification to the central server. This server lies in the middle of the I-Document environment and only stores the summarizations of each document, not the entire file. These highlights, or rather keywords, are stored for the duration of the user's network connection, and they are employed so other users can search through them and find a relevant document according to their search parameters. The automatic classification

cycle is defined by the client. Thus, if a user constantly modifies documents, he or she can set a short cycle time so that the keywords constantly sent to the database represent the latest version of each document. This makes I-Document practically real-time.

Set of Documents



Figure 1.2- 2: Classification of Local Documents

Once a classification is ongoing, users can proceed to search for needed documents and information. Clients simply choose their search criteria (author, document information, date, or file type) and input some keywords summarizing their search objective. Results are returned that match specific queries, and users can download these documents by just clicking on the file. A direct P2P connection is then established between the owner of the specific document and the seeker.

A final aspect of I-Document is its metadata optimization page. Users can click on one of their shared documents, view its metadata (data used in the classification that define it), and modify it to better classify the documents. Artificial intelligence has its limitations and document representations will not always be perfect. Thus, this metadata service gives clients an optional opportunity to fix any summarization errors.

I-Document is by no means the answer to existing problems with relevant information retrieval, and it is not the most complete solution considered throughout this semester's research. It simply provides a unique framework utilizing distributed computing. The framework can be easily modified and improved as new extraction theories and programs are developed. The following thesis demonstrates the current version of I-Document and the research behind its fabrication.

1.3 Thesis Organization

This thesis is divided into nine sections. The first is the topic introduction. The second section provides an analysis of the code complexity and project effort prior to any programming. This section presents an estimation of the function points, lines of code, and timeline for I-Document, as well as a review of its features in UML diagrams.

The third section describes the research behind the project's primary features. Examined areas include Information Extraction, Information Retrieval, text summarization, and metadata extraction. The section then provides a look into what was employed in the actual project.

The fourth section discusses the text summarization part of I-Document. It reviews the research behind keyword extraction, the system's use of summarization, and possible future improvements.

The fifth section describes metadata and how it works with I-Document. This portion of the thesis shows some features of the application.

The sixth section gives a technical overview of I-Document. It begins with a technology review, then moves on to the specific features in the application, and concludes with a system assessment. Pictures are also included to help readers better understand and visualize the product.

The seventh and eighth sections cover the details regarding competition and the I-Document company's business strategy. It also shows an analysis of the potential customer pool.

The final section of the thesis discusses conclusions regarding I-Document.

2.1 Code Estimation

Before beginning the code for I-Document, an effort to estimate the size, or the total scope, of the project was first required. This estimation is crucial for software projects to ensure that the ideas generated in brainstorming the vision do not require an excessive amount of programming that would necessitate a longer window of time than already established. The window to complete I-Document was only four months. Thus, the timeline estimation could not exceed four months or else the scope would have to be limited. Lacking size-estimation software, this approximation relied on a human, algorithmic approach to the quantify I-Document's size. This calculated exertion focuses on three main areas: function points, lines of code, and timeline.

2.1.1 Function Points

A function point is a synthetic measure that is often used at the beginning of a project because it is the most accurate way to think of size with limited development. Types of function points include inputs, outputs, inquiries, logical internal files, and external interface files. Each of these types is measured in terms of quantity and complexity, and then each is multiplied by an influence multiplier to properly adjust its count number to an accurate and applicable number. Below is a function point estimation for (1) the I-Document Web page and (2) the application. The Web page or portal provides information on I-Document and allows users to register and download the program. The application will include the product, Web services, and database coding. The application is the meat of the project while the Web page is just a release point.

2.1.1.1 Web Page Estimation

WEB PAGE	TYPE	COMPLEXITY
Login	Input	Low
Register new user	Input	Medium
Exception cases	Output	Medium
Download	Output	Medium
Static Pages (roughly 3-4)	Output	Low

Table 2.1.1.1- 1: Function point definition for the I-Document Portal

In this estimation, static pages like the homepage and the developer information page count as a single low complex output because they are simple in structure and programming. For database tables, one table is estimated in association with the Web site. This table concerns itself with user information. Having just a simple layout without messages and interfaces, they have a medium complexity level. Thus, the function points for the Web page are:

Program Unit	Low	Medium	High
Web inputs	$1 \times 3 = 3$	$1 \times 4 = 4$	$0 \times 6 = 0$
Web outputs	$1 \times 4 = 4$	$2 \times 5 = 10$	$0 \times 7 = 0$
Web queries	0	0	0
Database tables	0	$1 \times 10 = 10$	0
Total			31
Influence Mult.			1.05
Adjusted Total			32.55

Table 2.1.1.1- 2: Function point estimation for the I-Document Portal

The influence multiplier is an estimation based on a published list of several factors which includes heavy use, performance, online data entry, reusability, online update, and change. It is a value that ranges from 0.65 to 1.35. The value 1.05 is an assumed weight based on existing evidence.

2.1.1.2 Application Estimation

APPLICATION	TYPE	COMPLEXITY
Login	Input	Low
Select automation time interval	Input	Medium
Select shared directory	Input	Low
IP address	Output	Low
Start/Stop/Go offline commands	Input	Low
Status and server response	Output	Low
Search	Query	High
Search Criteria	Input	Medium
Download (P2P)	Input	High
P2P response	Output	Medium
Metadata Inspection	Input	High
Metadata Modification	Input	Medium
Metadata response	Output	Medium

Table 2.1.1.2- 1: Function point definition for the I-Document Application

For database tables (logical internal files), four tables are estimated to be utilized by the application. Two tables will be used for authorizing users, one for document information storage, and the final table for keywords. Again, having just a simple layout without messages and interfaces, they have a medium complexity level. For external interface files, Web services on the server side of operations are considered. Only one Web service is called, but the extent of the tasks influences a rating greater than one function point. Thus, Web services encourage an estimation of five functions on the server side at a medium complexity level. Table 2.1.1.2- 2 shows the function points for I-Document.

Program Unit	Low	Medium	High
Program inputs	3 x 3 = 9	3 x 4 = 12	2 x 6 = 12
Program outputs	2 x 4 = 12	2 x 5 = 10	0
Program queries	0	0	1 x 6 = 6
Database tables	0	4 x 10 = 40	0
External files	0	5 x 7 = 35	0
Total			136
Influence Mult.			1.25
Adjusted Total			170

Table 2.1.1.2- 2: Function point estimation for the I-Document Application

2.1.2 Lines of Code

This estimation tool takes the number of function points and multiplies them by the average number of code lines needed to implement those points in a specific computing language. With a preliminary value for lines of code, project managers can better assign their people and quantify the time of the coding process.

Program Unit	Tool	Lines/function point
Web inputs and outputs	Visual Studio	50 (code generation)
Application inputs, outputs, queries	Visual Studio	50
Database	Access (SQL)	10 (code generation)

Table 2.1.2- 1: Corresponding lines of code per function point

Utilizing the unadjusted values for function points:

$$\text{Web: } 21 \times 50 = 1050$$

$$\text{Application: } 96 \times 50 = 4800$$

$$\text{Database: } 4 \times 10 = 40$$

This amounts to 5,890 lines of code. Some of this code, however, may be automated by Visual Studio .Net.

2.1.3 Timeline

Given the lines of code, a fairly accurate timeline can be composed from existing industry averages. For an estimation of the timeline, 6,000 lines of code will be assumed to make the process easier. A slight overestimation, in addition, is a smart idea since most projects exceed their calculated timeline. On average, developers underestimate the time needed to complete a project by 20 – 30% [Dr. Kocur's 1.264 Lecture notes].

The nominal schedule for business products is used to estimate the timeline since all programmers involved have classes to complete along with this thesis project. Thus, the time to work on the project is limited. The other choices of shortest possible schedules and efficient schedules are not realistic assumptions. Given all computed values applied to the nominal schedule, it will take 3.6 schedule months and 5.4 man-months to complete the entire project. This falls within the time range.

2.1.4 Code Estimation Conclusions and Validity of Results

Looking at the timeline and lines of code, the project is within the set schedule and capacity, but the scope may have to be limited to complete the program in time. Thus, extensive functionality may have to wait until the next version.

In this exercise, the size and effort was estimated. Now while accepted industry procedures were employed, software estimation is difficult and not precise. Thus, there is almost a zero percent probability of hitting 5,890 lines of code or 3.6 schedule months from start to finish. Only twenty-five percent of large projects deliver on time [McConnell]. Given this fact, the programmers did not desire taking chances.

To ensure the project presentation date was hit (April 2, 2004) with a fully operational product, rapid development principles were used in fabrication process. The spiral model was used for development and consistent testing was maintained throughout the project. In addition, the scope was never allowed to creep beyond reach. Smart software practices

like these and the initial estimation helped lead to completing I-Document with a few days to spare.

2.2 UML Design

Using UML, or Unified Modeling Language, was the second main part of the project preparation. UML is just a modeling language that demonstrates how the processes of a system will flow and interact with a user. These state diagrams show details of user scenarios and behaviors in valid states. Utilizing UML helps developers and visionaries to clearly see the requirements, design, and direction of software products. It also speeds up the requirements process since project communication is improved through the introduction of visual diagrams.

Once the requirements were set and the product visions were mostly agreed upon, some state and activity diagrams were sketched and refined to assist project development. The resulting UMLs represent four main operations: the Web site's activity, the main application's functionality, the interaction with the server, and the P2P connection.

In the Web site activity diagram (Figure 2.2- 1), the framework is laid out for all functions available to an Internet user. A new visitor can first view information on I-Document and its production team. If the user then decides to download the product, he can register a new account. Successful registration gives access to the download page. Existing users simply log in with their existing name and password to access this

download page. Once the download page is reached, a user has reached the limits of the portal. The Web site's purpose is only to release information and the product.

In the application activity diagram (Figure 2.2- 2), the framework for the I-Document program is drawn. Upon starting the program, a user enters information regarding his name, password, classification cycle time, and directory of shared documents. If login is successful, the client has accessibility to the grid and is able to search through it. He chooses his criteria for searching (document type, metadata, and user), types some keywords, and then he has the option to download the retrieved documents from the query or construct another search. Instead of searching, the user can also choose to view the metadata from his documents and modify them. If a problem is encountered, he can view the help document to solve a problem with the application. Once a user is done with I-Document, he can cancel his connection, quit the entire application, or just keep it open.

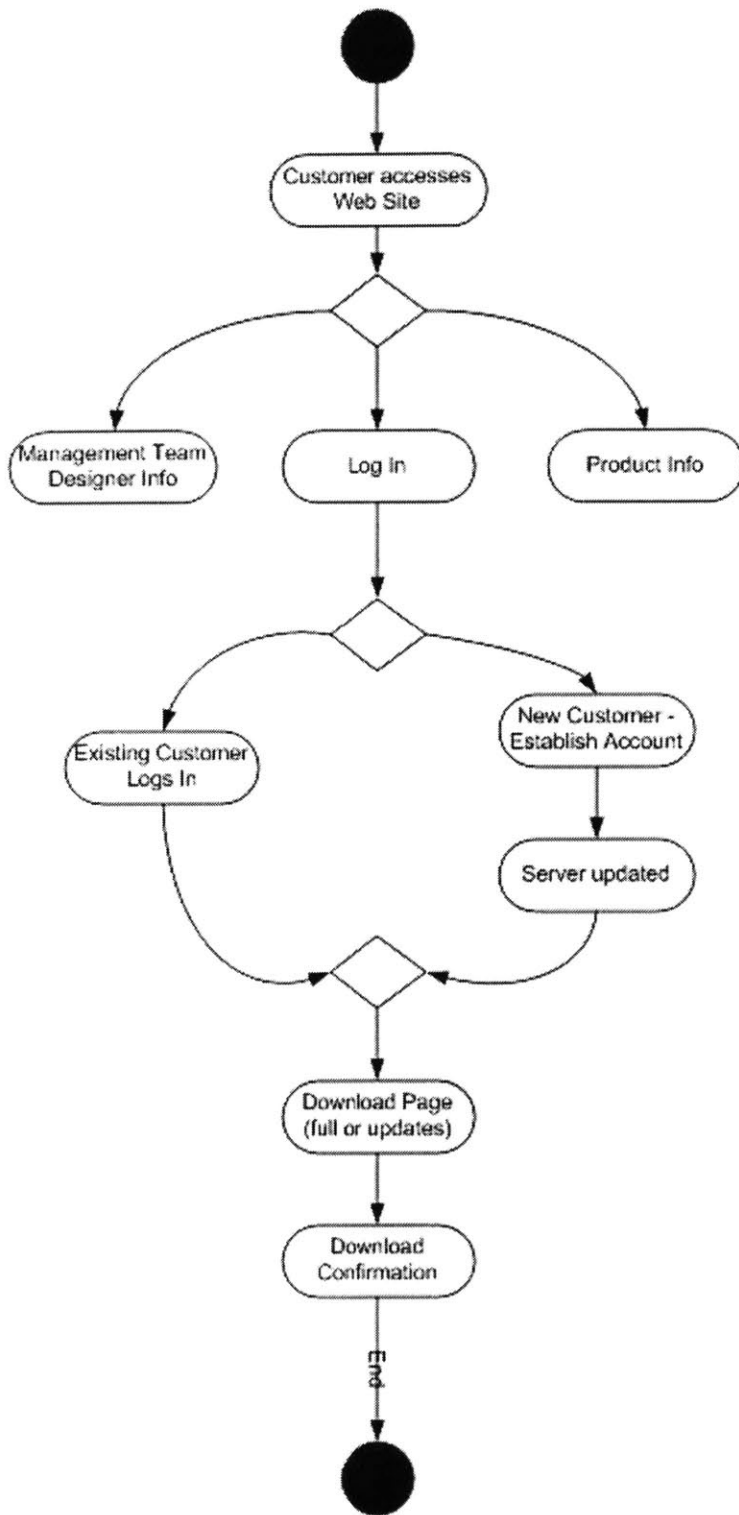
The client/server interaction diagram (Figure 2.2- 3) shows how the application communicates with the server. This diagram first demonstrates the log in procedure. Pressing the login button, the local application checks the client's shared folder. If there is a security or existence problem, the login will fail. If the folder checks out, the name and password will be sent to the server for a second test. With a name and password verification, the classification scheme will begin on the client side and keywords will be sent to the server according to the individually chosen cycle time. This classification

process is repeated automatically. The P2P listener also is initiated so other users can download documents from this client.

The other noticeable part of the client/server diagram (Figure 2.2- 3) is the search layout. Users are able to search through the I-Document grid even if they are not logged in to the system. This document service is free and open. When a user submits search information, the server takes this data and queries the database. Any relevant documents are returned to the client where the user can then decide if he wants to download any or all of those documents. If the files do not correspond to his search, the user can refine the search with different keywords and try again.

The final UML diagram (Figure 2.2- 4) demonstrates the P2P connectivity between two users. A user who is already logged in (left client) has his P2P listener activated. Another user (right client) hits this listener requesting a download, a download socket is chosen, and the document is sent. This relationship will only work if the owner of the document is successfully logged in to the system.

Together, the four diagrams outline the major features and direction of I-Document. Naturally, these were modified over time, but the overall scope did not change. Only a few of the specific details changed during I-Document's construction to get to the overall goal. Overall, the initial vision was maintained to meet the short time frame.



Web Site Activity Diagram: Simple download framework

Figure 2.2- 1: Web Site Activity Diagram

Application Activity Diagram

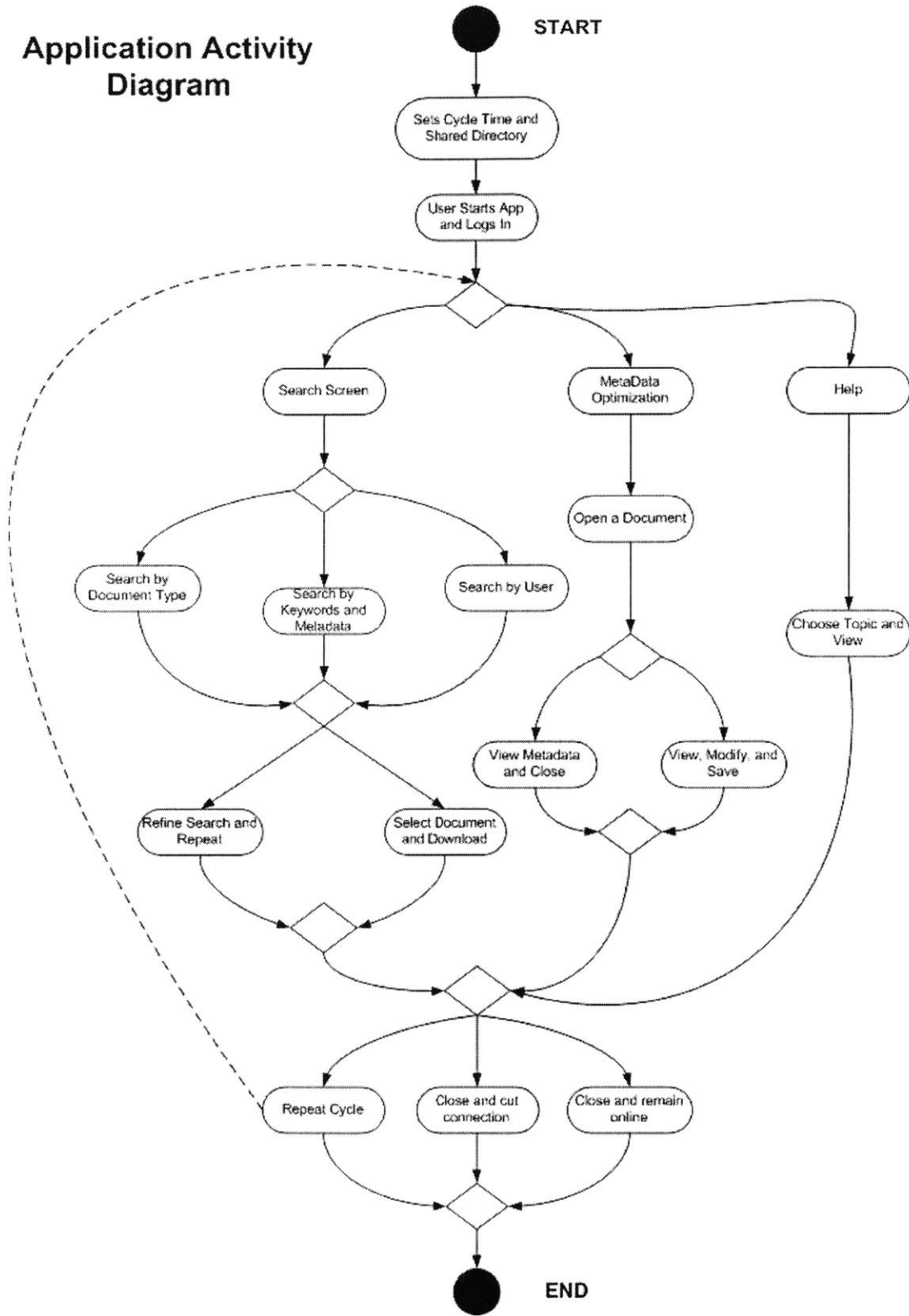


Figure 2.2- 2: Application Activity Diagram

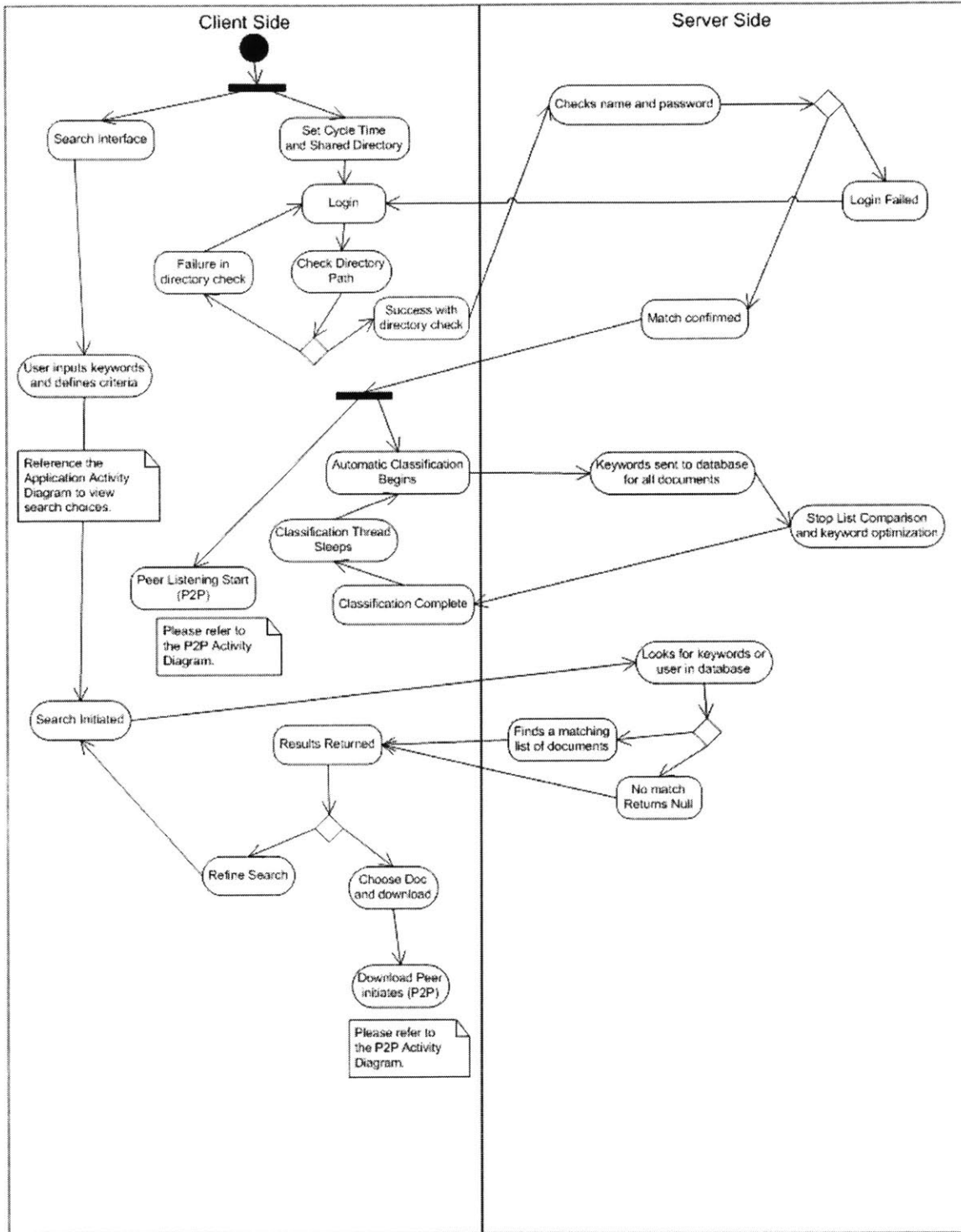


Figure 2.2- 3: Client/Server Activity Diagram

3.1 Classification System Design

Several months of research went into I-Document on how to handle searching and data mining. For the computer to automatically classify each document, it has to extract relevant data. Thus, from the raw documents on each client's computer, the classification system needs to extort valuable and structured information. For this structured data, three sources of information are gathered: file properties, metadata, and keywords. The first part of this search was the easiest. Extracting and storing file information as strings took only a few lines of code (this code is present under the GetExtension function in Appendix 1). This function stores certain pertinent aspects of every file like the creation date, length, and file type. The two other aspects of automatic classification, unfortunately, took much more time and research.

Before even attempting to extract metadata and keywords, the system layout of I-Document needed to be conceived. Obviously, searching is the heart of the application. There are two basic searching systems: Information Retrieval (IR) and Information Extraction (IE). Dating back to roughly forty years ago, Information Retrieval came around first in programming research. Today, it is one of the most important subjects in Computer Science. IR concerns itself with organizing documents, storing them, and then finding relevant documents according to a user's specific desire. Information Extraction just began to surface in the late 1980s, and it is defined as the process of analyzing and presenting information extracted from a file that is relevant to the user. Thus, IE operations tend to be more knowledge intensive and precise than IR operations. From the

definitions, the two systems seem similar. They both aim to return information that matches a user's search. However, these systems differ in how they should be utilized.

3.2 Information Retrieval Overview

An Information Retrieval System processes a set of electronic documents to find the one that best matches a user's need. More specifically, it looks for the documents that are meaningful according to a given query. The main components of IR are the user's information need, the request, a keyword-based query, objects stored on the main computer, and appropriate computer programs to execute the retrieval. In a typical case scenario, a user will formulate his request in natural language, but he will try to focus on key phrases. This request is then translated into a query using the main words. The query is sent to the correct machine and the matching results are returned to the user. Hopefully, these results are useful to the user. If they are not, the search is refined and executed again.

In terms of mathematics, the concept of IR is simply formulated. The classic model used in virtually all commercial IR systems is:

$$IR = \mathcal{R}(O, Q) \quad (\text{Equation 1})$$

In this formula, O represents the object base (set of documents on a computer which is searched through) and Q is the query. These are represented in this formula as vectors of numbers (mathematical objects). \mathcal{R} is defined as the relationship between these two objects. Using the concept of vectors to represent the documents and queries, one can use Vector Space Modeling as the fundamental IR system model.

Vector Space Modeling (VSM) is one of three classical IR models. The other two are Probabilistic IR (PIR) and Boolean IR (BIR). With VSM, objects ($O_1, O_2, O_3, O_4 \dots$) are certain distances in vector space from the query Q . Thus, any object and query is assigned a vector v of finite real numbers. These numbers, or weights, typically range from 0 to 1. As the distance d decreases between O and Q , the object becomes more relevant to the user's information necessity. As d increases, the object is less applicable. VSM is the most basic of the classical IR models, and it was soon built upon using probability.

The Probabilistic Information Retrieval model works in conjunction with VSM. PIR was "first pioneered in the UK at Cambridge University and London's City University during the 1970's and 1980's"¹. The idea behind PIR is to use feedback from the user to further estimate the degree of document relevance. This is another angle at measuring the distance d . For a real world example, consider the PageRank system employed by many Internet search engines. These systems increase the relevance of a certain page as more users link their pages to that Web page. This PageRank system is explained in more detail in Section 7.2. For documents, the criteria for weighting cannot rely on links. Thus, it instead uses measures like term frequency, document length, and user feedback. To work properly, PIR applies Bayes' Theorem to rank the relevance of documents according to a query.

¹ <http://www.conceptsearching.com/conceptFAQ.htm#a2>

Bayes' Theorem is the most significant theory regarding conditional probability to date. It was published posthumously in Thomas Bayes' masterwork, "An Essay toward Solving a Problem in the Doctrine of Chances" (1764). The theory "relates the 'direct' probability of a hypothesis conditional on a given body of data, $P_E(H)$, to the 'inverse' probability of the data conditional on the hypothesis, $P_H(E)$ "². Bayes' Theorem is stated as:

$$P_E(H) = [P(H)/P(E)] P_H(E) \quad (\text{Equation 2})$$

Or written in a different form:

$$P(A | B) = \frac{P(B | A) P(A)}{P(B)}$$

Using natural language, the second form says: $P(A | B)$, the probability that event A will occur given that B has already occurred, is equal to the probability of A occurring given B has occurred, multiplied by A 's occurrence probability, all divided by the probability of event B . Programmers use this theorem in PIR to ascertain the weight of query terms and the relevance of those terms in a given set of documents.

The third classical IR model is Boolean Information Retrieval. This was the first adopted model and is the most widely implemented. It is based on Boolean Logic which was originally developed by George Boole in the 1800s. It uses primarily the AND, NOT, and OR operators to conduct retrievals, and the model returns all results without weighting relevance. Unlike PIR, it does not employ a filtering or probability mechanism to limit the results. Thus, in a large database, the results can be overwhelming.

² <http://plato.stanford.edu/entries/bayes-theorem/>

Beyond Vector Space Modeling, Probability Information Retrieval and Boolean Information Retrieval, there lay many other unconventional models. First are non-classical models which include Information Logic IR, Situation Theory IR, and Interaction IR. Information Logic relies on a logical inference process in retrieval. Situation Theory proposes an Information Calculus for IR. Finally, the interaction model focuses on the interaction between the query and interconnected documents. The second group involves alternative models. This group includes the Fuzzy Model, the Cluster Model, and the Latent Semantic Index Model. The Fuzzy Model is like BIR except it represents a document with a fuzzy set of terms (a set of terms and their function). The Cluster Model is just like VIR, but it groups documents together to speed up the retrieval process. The Latent Semantic Index Model (LSI) is also like VIR, but it weighs the vectors differently. LSI uses artificial concepts to better capture the general meaning of documents. The third and final group is Artificial Intelligence. This method is code intensive and is very difficult to implement. Essentially, it uses Knowledge Bases (KB) and Natural Language Processing (NLP) to enhance an IR system. AI is not a system that is easy to create from scratch in three months, especially with programmers lacking experience with Artificial Intelligence. Thus, this model was immediately out of the project scope.

With these models defined and investigated, a general idea of the type of Information Retrieval System that could be implemented in three months was assembled. But IR is only useful in the database searching portion of the I-Document project. An Information

Extraction System is better suited to search through and classify documents. Thus, IE is a more pertinent and important subject of investigation given that I-Document revolves around the automatic classification of documents.

3.3 Information Extraction Overview

With conventional IR systems concerned with a basic keyword search and ranked retrieval, more sophisticated tools are needed to automatically analyze unstructured documents. The effort to find these precise tools is found in Information Extraction. IE Systems analyze documents and present the appropriate information to a user's request.

During the eighties, the first attempts at these systems began in the financial field.

Companies created systems like ATRANS, JASPER, and SCISOR to extract facts from databases that related to a certain company's earnings and financial history. These systems were a huge step, but their major shortcoming was not being able to adapt to new scenarios. Today, Information Extraction Systems are developed for a broad spectrum of e-document intensive companies, governments, and academic communities. These systems range in their framework, but most follow two rudimentary approaches: knowledge or learning.

When designing an IE System, the two approaches to choose from are the Knowledge Engineering Approach and the Learning Approach. The Knowledge Approach has a solid, unbreakable set of rules for marking and extracting information. The creator of the system establishes these rules. Thus, the system's success relies entirely on the expert

knowledge of the programmers. They set the system's rules for extraction, and continuously refine it based on test cases and their own intuition.

The Learning Approach is the other path programmers can follow when designing an IE System. This approach has a framework with master rules, but the system adapts its extraction rules from interaction with users. Thus, this approach begins like the Knowledge Engineering Approach, but it learns and improves itself without expert interaction. Currently, a hybrid system is in development, but progress is slow.

Once an approach is decided upon, the designers then need to clarify methods of extraction. These methods pertain to data mining and text processing, and they will be discussed in the fourth section of this thesis.

3.4 The I-Document System

For the I-Document system, one type of model does not satisfy the design specifications since there are multiple searching aspects in the program and service. I-Document needs an Information Retrieval System to find relevant documents according to a query and return them to the searcher. On the other hand, it also needs an Information Extraction System to analyze documents and extract relevant information so that those queries are successful. Thus, an IR System is utilized for the database search and a simplified IE System for the automatic classification. This type of split in the design is reasonable because the classification extracts data from documents while the database search retrieves documents. This split also gives the students within the project the

opportunities to learn about each implementation and try to apply their fundamental concepts.

For the Information Retrieval System, I-Document uses a Boolean model. This is the most widely adopted model, and is easier to code in conjunction with Microsoft Access. Unfortunately, the downside to using BIR is that it retrieves all documents that fit a query. Currently, there is no filtering of results or ranking system employing Bayes' Theorem to stress more relevant documents. This is an area that should be changed for future versions of the program, especially as the base of documents grows.

For the Information Extraction System, I-Document uses the Knowledge Engineering Approach. This approach is simplified because it only focuses on extraction procedures. The presentation of relevant information pertaining to a user's request is left for the IR System. The Learning Approach is not employed because it involves too large of an effort for a three month project. Designing a learning system coupled with artificial intelligence is not in the scope of the original design, and programming it would have led to an extended time frame.

With the Knowledge Engineering Approach, I-Document's extraction techniques rely entirely on the programmers' expertise. For data extraction, storing keywords and primary phrases is sufficient in describing a document. Thus, in the analysis process, a system with text summarization along with an investigation of file properties is used.

With this approach, a system was successfully designed, guided by the original concepts and theories, to automatically classify documents.

4.1 Text Summarization Overview

A significant part of classifying documents for the project was a text summarization. I-Document utilizes a simple algorithm to search the full text of a document, extract keywords, and only keep those keywords that are not common in the English language. Thus, this is an effort to get a short list of words that describe the theme of each paper. Fundamentally, the algorithm tries to get at a reduced representation of the content. There are several ways to handle this part of the project. Focusing on framework, the application takes an uncomplicated approach to the algorithm just to ensure it works before the deadline.

4.2 Summarization Research

To begin the summarization code, an investigation of the layout of documents and their context is needed. The performance of the text summarization depends on term selection. Thus, a short, precise list of terms is required to represent the overall text. The fewer the terms extracted, the less intensive the application is on time and memory. With this selection goal in mind, selection research then orientates itself around two types of words in a document: tokens and types.

Tokens are running words of a document. They occur more than once and typically represent the topic of a paper. These words are not common in the English language. Thus, their frequent use in a document indicates their importance in divulging its purpose. These are the most desired keywords in the extraction. Types, on the other

hand, are individually recurring words that are common in the English language (i.e. “the”). They have no value in signifying a document’s primary message. I-Document’s algorithm will be looking for frequent words, so these will be picked up over and over again. Therefore, a stop list was needed to halt these types from being stored as keywords.

A stop list is used by virtually every text retrieval system and contains common words in a language. These words are expected to be used multiple times in speech. When a search is conducted, the computer system refers to this list to see if a frequent word should be used in classifying a text. Term frequency is computed through the following formula:

$$F_{ki} = n_{ki} / N_i \quad (\text{Equation 3})$$

In this frequency formula, F is the relative term frequency, n is the number of occurrences of term t , and N is the total number of terms in a document. The higher the frequency, the more common the term is in that particular text. Very common words provide little meaning. While “the” may occur in a text the most, it will not tell you anything about the point of a document. Thus, the most common words in the English language are included in the stop list. A less common token, however, has deeper meaning.

A systemic way to retrieve tokens is outlined in the following steps:

1. Exclude all stop words (types)
2. Stem the rest of the words to get at their linguistic roots (See Section 4.4)
3. Compute these words’ frequencies
4. Rank the words according to term frequency

5. Exclude very high and very low frequency terms
 - Have thresholds in place to determine high and low limits
 - Remaining terms are document identifiers
6. Document identifiers are the desired tokens

Once a stop list and term frequency method is in place, a processing environment is then required.

There are two basic types of processing environments for IE Systems: Deep Text Processing (DTP) and Shallow Text Processing (STP). DTP processes all interpretations and grammatical relations in a document. It is highly complex and returns a large number of keywords and phrases. Being so complicated, it is code intensive. It is also, at times, not necessary for many summarization systems due to the time and cost it takes to implement. STP is less complete than DTP, but it is simpler and less time-consuming. It does not look into the relationships of words, but an STP system does retrieve an efficient number of keywords. Often, these engines prove to be just as accurate as DTP systems. Thus, from the start of programming, efforts were focused on an STP engine to drive the classification.

4.3 I-Document's Text Summarization

Following the fundamentals employed by Shallow Text Processing Environments, a basic system for extracting keywords was designed to represent shared documents. The code for this summarization algorithm currently only works for a txt document, but it can be easily extended into Office documents and text files from different operating environments. Once the code regarding opening and reading a file is discovered, the

algorithm can be applied. The algorithm is not advanced, but it accomplishes the project goal for extracting keywords.

The algorithm opens up a document and looks at the frequency of each word in that text. Left with a ranked frequency list, the system first automatically eliminates words that are three letters or less. This is a crude method for extracting keywords, but it is higher recommended by summarization texts. Very rarely are keywords two or three letters long, and using longer keywords is acceptable for the purpose of this application. Once the short words are eliminated, the others are compared to two stop lists. These lists will eliminate any common words.

For I-Document, two stop lists were created to distribute the searching process. One is located on the local client side. It is a very small list that contains about thirty-five extremely common words like “the”, “that”, “she”, “with”, “yes”, and “and”. The second list is located on the server. This server list currently contains seven hundred and ninety words, and it can be easily modified while the local stop list cannot. Splitting the stop list is not necessary, but it demonstrates the power of Web services and distributive computing.

Splitting the stop list also carries a huge advantage. The database operator can modify the primary stop list on the server without having to change the I-Document application code. The operator has access to this second list, and he can add or subtract words as time progresses. The stop list words were collected from a page containing the thousand

most common words in the English language. Appendix 2 contains the complete stop list used in I-Document. Further research and changing speech patterns may prove that this list needs to be modified. This is why the main list is left accessible to authorized administrators.

Once the most common words are eliminated using the local stop list and then the server list, the system looks at the frequency of the remaining words. These words are the keywords, and the system stores the four most frequent tokens of this group. This number can be extended to as many as are useful.

4.4 Improvement of the System

I-Document's method of text summarization is not perfect. It can be enlarged and made more complex to optimize classification techniques. Looking into several possible ways of extending the application, some improvements were seriously considered.

Unfortunately, the time and computing resources to implement these improvements was not extensive enough. These possible improvements range from searched documents, languages, stemming, query expansion, and storage.

The first area of improvement is the searched document base. Currently, I-Document only summarizes the text of txt documents. This is not enough. Including all Office documents would take some time, but the benefits would be great. Analysis of Word and PowerPoint documents could be performed exactly like txt documents. With Excel spreadsheets, the focus would be on word extraction and not number frequency. Some

problems may arise, though, if the spreadsheet contained no words. Thus, some type of word check would first need to be in place. Unfortunately, too many unexplainable errors were encountered while opening multiple Office applications. I-Document was not solid enough with these ideas. It was suggested to examine this problem with Office 2003, but the cost of the software was not in the scope of the project.

Once Office is integrated, other files like Adobe and those from other text processing applications could be adopted. With some file types, ignoring a text summarization unfortunately would be forced because of the complicated context. For example, searching through the code of computer programs would be very difficult. A different stop list would have to be created to include system words and the specific code summarization algorithm would be limited to investigating and ranking words regarded as programmer comments. Unfortunately, programmers rarely write comments to outline their entire code. This effort would ultimately prove to be a waste of effort and time.

The second area of improvement involves languages. Currently, I-Document is only geared toward English. While English speakers are the largest group of Internet users, the number of non-English speaking Internet users is growing rapidly. It is estimated that by 2005, non-native English speakers will represent seventy percent of the online community. The problem with using a Shallow Text Processing System is that it is specifically tailored for one single language. It is very difficult to move this environment to multilingualism. The main effort to search through documents in another language is called Cross-Language Information Retrieval (CLIR).

CLIR efforts are growing, but it is very difficult to switch meanings from one language to another. Often, a language expert needs to step in to verify translations for current systems. In CLIR, two architectures have been explored and coded. The first is a query translation system. This entails translating a query into the language of the documents. Thus, to implement this feature, it is necessary to modify the query code of I-Document and the words in the stop list while keeping the search algorithm unchanged. The second architecture is a document translation system. This method takes original documents and translates them into the system's primary language. This method is over the top for I-Document's purpose. It does not need to translate a document, extract its English keywords (which may or may not have been translated correctly), and store those tokens. Query translation would be easier and more efficient because it only translates a few words. Given that CLIR would require significant code modifications and the storage space for a multilingual dictionary, it is not foreseen as a feature of I-Document in the near future, but it is an ambitious thought.

The third area of improvement is stemming. Stemming is the process of reducing words to their roots. This feature into would work very well with text summarization and searching for documents. For text summarization, the system would take all words, stem them, and compare them to a stop list. For searching, I-Document would take a user's specific query and stem them as well to match the stemmed words in the database. Stemming will create a more powerful search term and better query results. The best known stemming algorithm is Porter's Algorithm. It was first described in 1980 and has

been widely used and adapted ever since its inception. There are even versions of it on the Internet available in multiple computing languages, including C# (a version is available at the Web site: <http://www.tartarus.org/~martin/PorterStemmer/csharp2.txt>). Putting this into the application would involve a simple code modification as long as the available code is correct.

The fourth area of improvement is query expansion. This extension typically uses a thesaurus to discover the relationship between words. Thus, if a user types in a query that does not match keywords in the database, but it is related to certain keywords, the query expansion system could pull up those related documents instead of retrieving a null set. The problem with this feature is that it requires the storage of a thesaurus and the construction of a relational system. One common way around this problem is using WordNet. WordNet is a Princeton University project that organizes “English nouns, verbs, adjectives and adverbs...into synonym sets, each representing one underlying lexical concept”³. Using these sets, called synsets, the query and possibly the text summarization portion of I-Document could be extended.

The fifth area of improvement is broadening storage. I-Document only works with Microsoft Access and SQL Server on individual laptops. Thus, space and computing power is limited. Pushing this onto a server (or servers) would allow an increase in the table size and the utilization of greater processing power. The system would then have the ability to store more keywords and whole key phrases. It could also work with multiple stop lists according to the document type. With enough storage space,

³ <http://www.cogsci.princeton.edu/~wn/>

dictionaries or thesauri could be included on the server to expand queries and summarizations. Currently, the I-Document code is only 10Mb, so there is much more room for improvement.

4.5 Summarization Conclusion

There are many ways text mining and summarization can be improved. This is why Natural Language Processing is such a hot field in computer science. It is a relatively new field that has enormous potential. This section's review just focuses on five areas that are potential additions to I-Document. The problem of parsing a document is not an easy one to answer. To generate a highly accurate full text understanding system is seen as impossible with existing technology. I-Document is a start. It utilizes light-weight linguistic analysis tools that are sufficient for extraction. This project was unfortunately limited by money, hardware, and time that hindered the realization of a complex summarization system.

5.1 Metadata Introduction

Extracting metadata from files is the final part of the application's classification scheme. Metadata is defined as information about information or data about data. For I-Document's purposes, it is user defined information describing each document in a few keywords. Since I-Document only uses keywords and phrases to represent entire documents, using metadata is a vital piece to the software. So far, the system has automatically classified each document. Now, it will utilize each client to define their own documents to better classify them.

Metadata is quite young when compared to the age of Information Retrieval and Information Extraction. Metadata began in 1995 with PICS, the Platform for Internet Content Selection. Then, the Dublin Core Metadata Initiative (DCMI) was created in March 1995 in Dublin, Ohio at a joint workshop between people from the NCSA and the OCLC. At this original workshop, more than fifty people described the beginnings of a core set of semantics to categorize the Web. Adopting these metadata standards would make searching and retrieval operations much easier and efficient. Since 1995, DCMI has held several workshops across the world to optimize those standards and promote the widespread adoption of metadata-based documents and retrieval systems.

5.2 Metadata Decision and Implementation

A metadata search function was designed and coded into the Web service because its benefits significantly added to the overall classification. This code was installed on the

client-side Web service with the rest of the classification functions. Given the right file, the service will extract certain metadata items and store them in the database. The search is currently set for Microsoft Office documents. Office has a metadata feature that allows users to write in some keywords into their document properties. To access this facet by hand, users can click “File” and then the “Properties” when an Office document is open. By clicking the “Summary” tab, a user can see the metadata types offered. They range from author and title to company and comments. Users can input what they need to describe particular document and save the information. There is also a custom feature that allows users to save more specific and random metadata information like department, language, and publisher.

To first get metadata extraction working, the programming was researched and a preliminary piece of code was found at Microsoft’s support site (<http://support.microsoft.com>). This code was heavily modified to work with I-Document. A large portion of this resulting code is included in Appendix 1. The function currently grabs the title, author, subject, company, keywords, and comments from Word documents. I did not feel the other metadata items (regular and custom) had any value with classifying documents. They were just overkill.

Unfortunately, this metadata option does not work too well with Office XP as it stands. To solve this problem, Primary Interop Assemblies (PIAs) must be downloaded and installed to give necessary controls to ASP.NET. PIAs allow ASP.NET to access the metadata in Office XP. I-Document then must have launch permissions for Office

applications to enable it to open and inspect Office documents. With these steps in place, the Web service could now open and inspect metadata properties. Office 2003 is supposed to have these features already set, but as stated before, the project was not granted the necessary funds for this software.

To enable the user to view and modify his metadata, a tab (Figure 5.2- 1) was added on the client-side application to change the metadata the Web service is extracting. The setup is self-explanatory. Users can open a file through I-Document and its metadata will appear in a listbox. The user then chooses to revise these items and save the changes. Through utilizing the user's information, this tab becomes an effort to get them involved with the classification to better understand metadata and ultimately improve the documents' representations in the database.

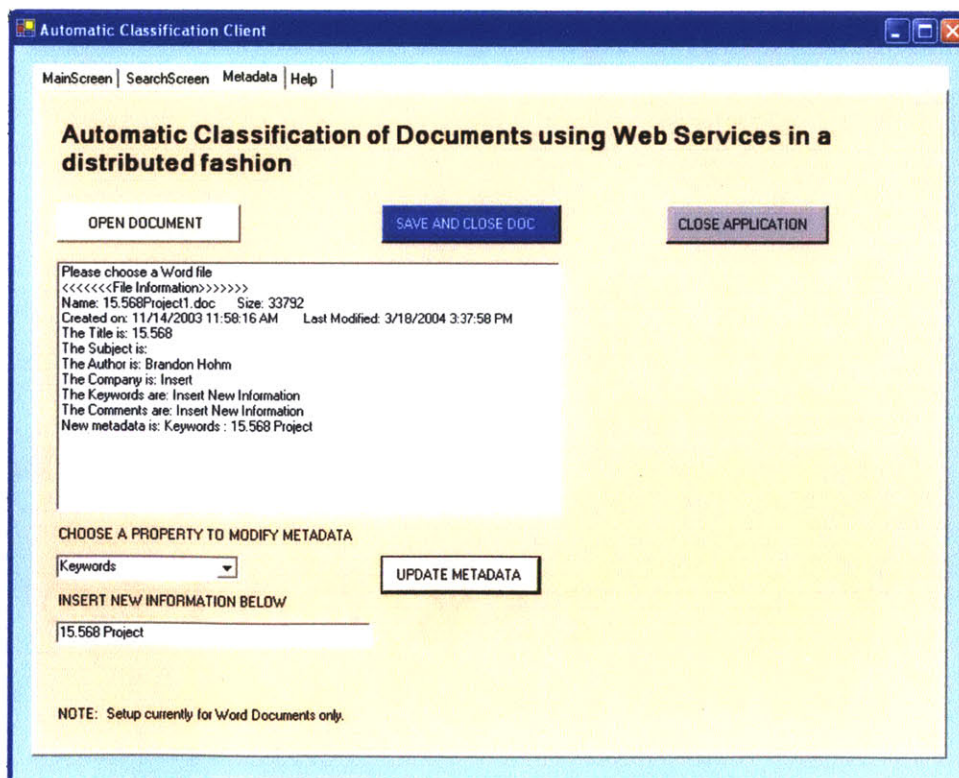


Figure 5.2- 1: I-Document Metadata Tab

5.3 Metadata Extension

Extending metadata will be a challenge, but hopefully as DCMI promotes its standards, more and more applications will utilize metadata to describe documents. Attempts were made to code something for Adobe and txt files, but too many errors and time constraints were encountered. An easy fix to this problem is allowing the user to describe every document himself and store this information. This method is commonly used for online libraries that accept documents from external clients. Users submit their paper with basic descriptions. These descriptions are often translated into XML which imposes needed structural constraints. While this would be an interesting solution, it would not hold up in I-Document's environment. Classification is continuously performed, and records are erased when users log off. Thus, clients would have to input and save this metadata information every classification cycle. This would be an unnecessary hassle. The current method works behind the scenes with optional user involvement.

Technical Overview of I-Document

6.1 I-Document

Employing Information Extraction and Information Retrieval techniques, along with mining file properties, text keywords, and metadata, I-Document is able to automatically classify documents and retrieve other users' files based on specific queries in a P2P environment. In this section, a review is given of the main technology behind the application and specific descriptions of each piece of the software.

6.2 Underlying Technology

I-Document was primarily created in Microsoft Visual Studio .Net 2003. With all members of the project group just completing a class in .Net and C#, this platform was the obvious choice. There were also secondary reasons for choosing this computing language and the .Net framework. .Net comes with many benefits and functionalities that compliment I-Document's visions and goals.

6.2.1 .NET and ASP.NET

Microsoft's .Net framework "provides a highly productive, standards-based, enterprise-ready, multilanguage environment that simplifies application development, enables developers to make use of their existing skill set, facilitates integration with existing software, and eases the challenges of deploying and operating Internet-scale

applications”⁴. This initiative is very broad and revolutionary. Its improved features include:

- High scalability
- Support for all CLR (common language runtime) types
- Embrace of XML and SOAP
- Rich object model enabling several functionalities
- Easy language integration
- Simple deployment
- Easy interface with existing software
- Greater developer control

Along with these features, the greatest advantage to using the framework was its ASP.NET environment. This environment is a tremendous advancement in Web-based development. It allows programs to communicate over the Internet using SOAP. SOAP, or “*Simple Object Access Protocol*, [is] a lightweight XML-based messaging protocol used to encode the information in Web service request and response messages before sending them over a network”⁵. By simply referencing a Web service in their program, developers can link a local application to any insecure service (or secure if the application is granted permission). .Net does all the rest. This is the feature that was needed to make I-Document work efficiently. The ease of programming a Web service was the biggest draw to using the .Net framework. The next major technical feature was the establishment of a peer-to-peer environment.

6.2.2 Peer-To-Peer Computing

P2P computing is a type of distributed computing. Computers, often low-power clients, are linked to aggregate processing power. Essentially, a peer-to-peer network is a loose, dynamic network of clients. This type of computing is still in its infancy. Yet, with its

⁴ <http://msdn.microsoft.com/vstudio/productinfo/whitepapers/default.aspx>

⁵ <http://sbc.webopedia.com/TERM/S/SOAP.html>

revolutionary methods of communication, collaboration, and power distribution, it was an obvious choice for I-Document's sharing environment.

There are two models of peer-to-peer computing. The first model is the pure model. This environment enables all users to have the same capability and share responsibility. This model is represented by the red arrows in the Figure 6.2.2- 1. The second model, or the hybrid model, establishes a middleman to facilitate interaction between peers. This model is represented with black arrows. I-Document utilizes the second model.

A server lies in the middle of the grid environment to regulate usage. This server authorizes users, stores indexes, and contains the searching framework. Downloads, on the other hand, follow more of the pure path. Using IP addresses, port, and file path, clients connect directly to each other to send and receive documents.

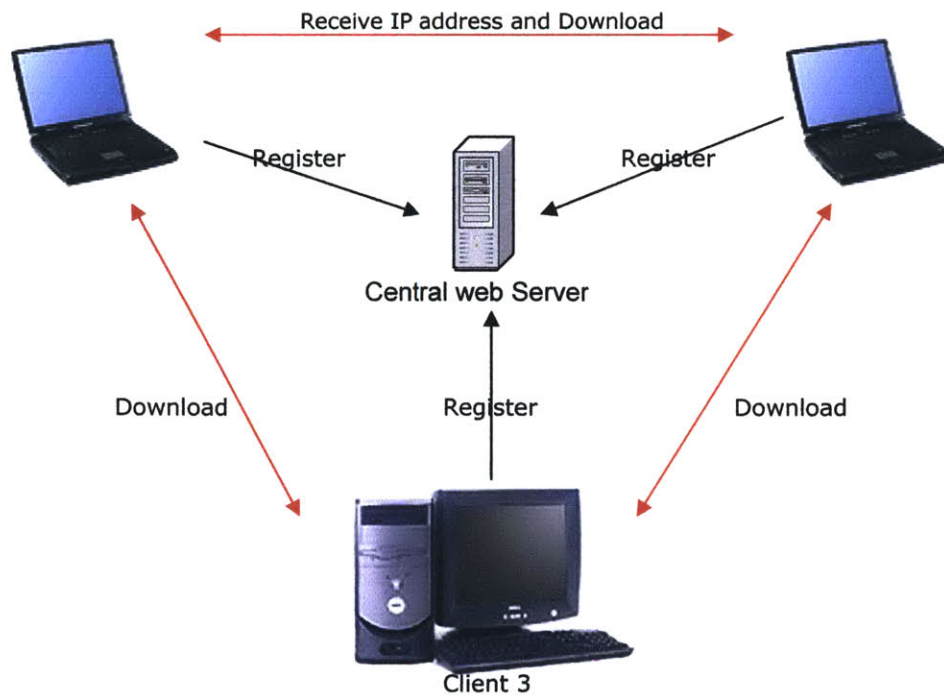


Figure 6.2.2- 1: I-Document's P2P Computing Environment

Utilizing this type of environment creates many advantages over other types of sharing, storage, and searching applications. The competitor comparison will be covered in the seventh section.

6.2.3 Web Services

A decision was made early in development to use two Web services in the I-Document application to demonstrate its power. Web services are said to be main element in the next major IT strategy. Fundamentally, Web services are constructed on the Internet and use standard communication protocols to enable communication between multiple applications and devices. These standards dramatically simplify passing information and distributing computations between multiple machines. While not a new technology, Web services are gaining momentum in industry as they are being increasingly used enhance existing IT platforms.

The architecture of Web services has three basic layers. The overlying layer consists of software standards (WSDL, UDDI, and XML) and communication protocols (SOAP, HTTP, TCP/IP). By using these commonalities, different applications can interact and work together. The middle layer is called the service grid. This consists of all the utilities that provide the functioning of Web services. Thus, tools include security utilities, performance assessments, monitoring services, messaging utilities, registries, knowledge management utilities, and reliability provisions. The final layer (the local layer) is the application service. These services support the processes of programmed

Web applications. In I-Document, the application service is what the client sees and uses on a daily basis.

The promise of Web services is intriguing. They allow organizations to move from proprietary information systems to a more open network. They also allow once incompatible systems to now work together in unison. The advantages of shared services are reducing operation costs and enhancing business strategies.

6.2.4 Microsoft Office

Several applications in Microsoft Office are used with I-Document. First, Microsoft Access stores user and document information. SQL Server is also used in conjunction with the portal to register new users. For other programs, I-Document uses several Office documents in the metadata extraction operation. With the right code, users can extract certain origination information from all types of files, but with Office XP and Office 2003, there are ways to extract and modify Office document properties using Web applications. Unfortunately, Office XP does not come with the correct assemblies to allow ASP.NET to access these properties, as discussed before in Section 5.2. A simple download and installation of these Primary Interop Assemblies (PIAs), nonetheless, solves this problem (download free from <http://support.microsoft.com/default.aspx?scid=kb;EN-US;328912>). Microsoft Office 2003 already includes these assemblies.

6.3 Pieces of I-Document

I-Document is split into three basic parts: Web services, a Web application, and a database.

6.3.1 I-Document Web Services

Web services are the heart of I-Document. There are two services: a central service on the server and a local service on the client computer. First, regarding the central service (Figure 6.3.1- 1), this Web service is called whenever a client desires communication with the server. Thus, it works hand-in-hand with the database of clients and document keywords. With this role, the central service verifies users, stores keywords, filters common words from keywords, deletes/adds users, modifies users' indexes, provides download (P2P connectivity) information, and searches through the document keyword table. These Web methods are available for any application to freely use. All they need to do is reference the Web service. For example, to check if a machine is in the database, an application would need to set this central service as a Web reference, and pass in a machine name and password under the `isNewMachine` Web method. Since I-Document is academically oriented, free access to information is encouraged. Thus, the service is accessible, but not the sensitive information like passwords and a list of registered usernames.

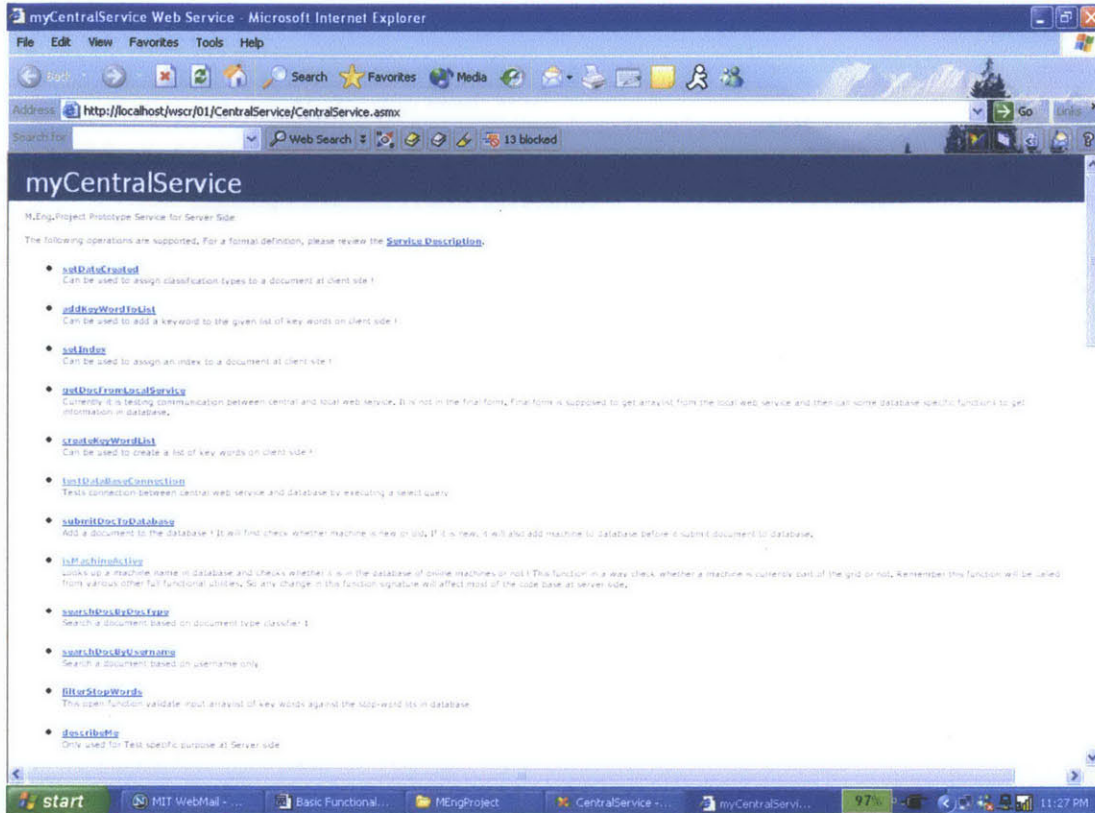


Figure 6.3.1- 1: Central Web Service

The second Web service is the local service (Figure 6.3.1- 2). Unlike the central service which is stored on the server and freely accessible via the Web, the local service is downloaded and utilized by individual client side machines. The most important aspect of this local service is the classifying Web method. Utilizing this method, clients classify their shared documents using their own computing power. The service inspects each file in the specified folder and stores its file type, creation date, full name, and author. Then, for certain text documents, the service examines the metadata and performs a full-text search, extracting keywords. All of these pieces of information are sent to the central service, where it is further analyzed and stored. The other Web methods of the local service operate as a bridge between the I-Document application and the central service.

The other Web methods on the local service simply pass on information to the server. This information includes machine name, password, and search criteria. I-Document takes typed information from the user, passes it through the local service and onto the central service where the required actions are performed, and the response is sent back to the local service and the application. For example, in searching for all documents under a certain username, the user inputs the name into I-Document and submits it. This text is sent to the local service and then onto the central service. The central service inspects the database, retrieves all documents under the specified username, and returns this dataset to the local service. The local service takes this dataset and returns it, as is, to the client application. The local service seems unnecessary when describing these other methods, but it was created for the classification algorithm and its functions. There was no desire to bog down the server with this time-intensive operation. Distributive computing carries many more advantages in the I-Document environment than basing the classifications on the server.

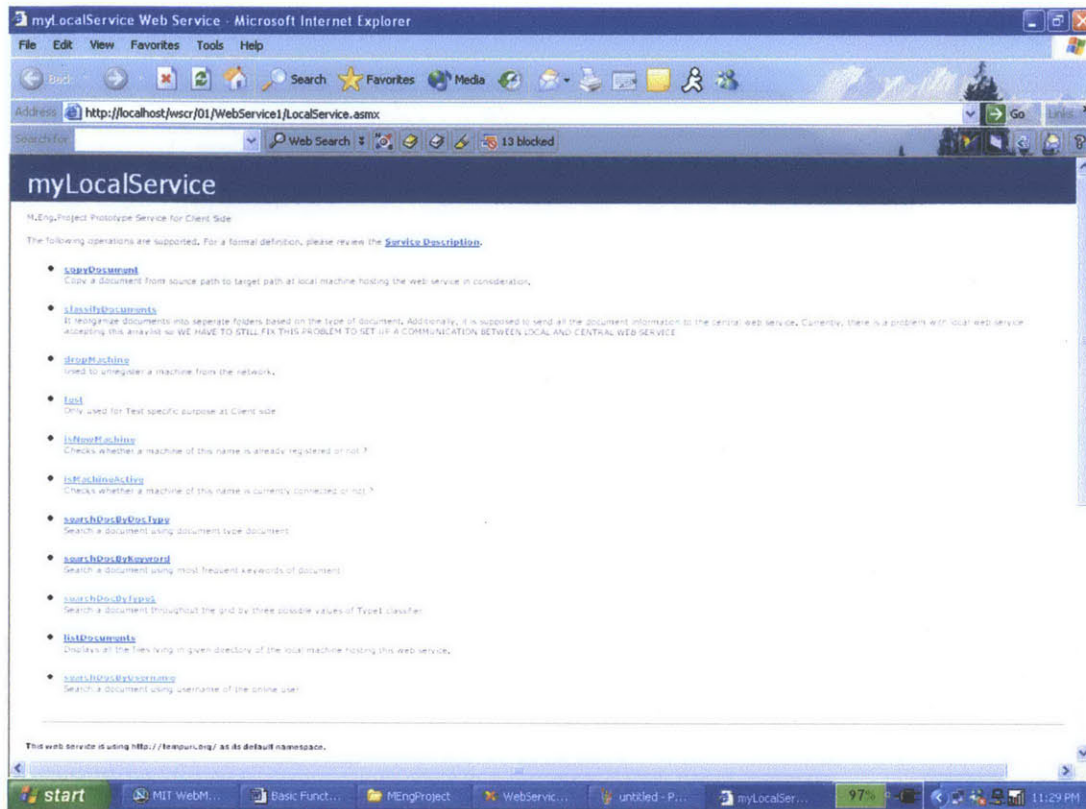


Figure 6.3.1- 2: Local Web Service

6.3.2 Web Application

The Web application (Figure 6.3.2- 1) works on each client computer and connects the user to the Web services listed in the above section. To activate the application, the user first sets the cycle time and directory. The cycle time is the time interval between each automatic classification. Obviously, with a smaller time interval, keywords sent to the database are more up-to-date or real-time. The shared directory is a folder of documents that the client wants to share with the I-Document community. With this information, the user then inputs her username and password, and connects to the service. If any of the above information is incorrect (wrong folder, incorrect password, etc.), the application will not connect. Upon connection, the classification cycle time begins, and the asynchronous P2P listener is initiated. This means that documents are continuously

classified and stored, and other users can download these documents via a P2P connection. The other aspects of the application are available even if the user unsuccessfully logs on to the I-Document network.

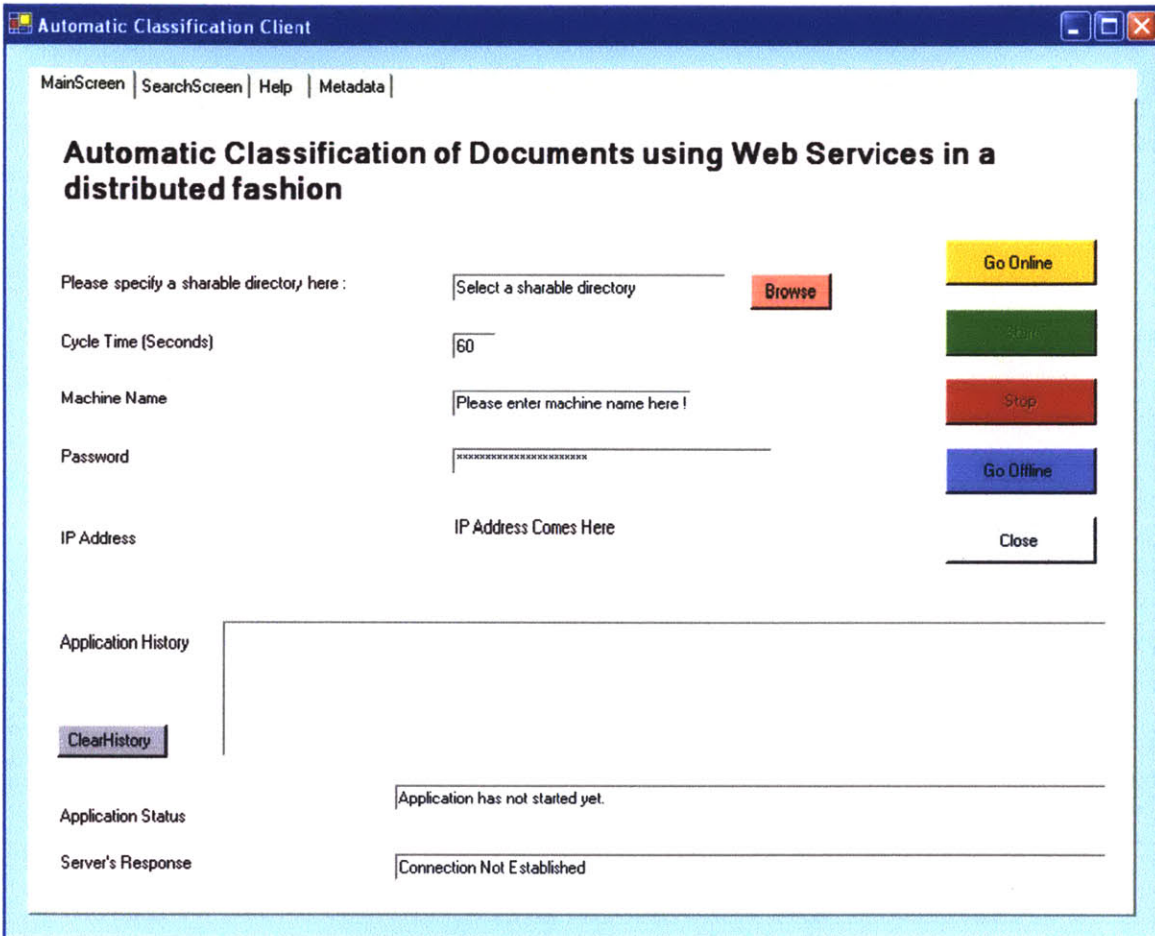


Figure 6.3.2- 1: Main Web Application Picture

The other aspects of the application are the search capability, help tab, and metadata optimization. On the search tab (Figure 6.3.2- 2), users can search for documents according to their chosen criteria. Users are able to find documents via their type, keywords, date, or owner. Results are returned from the database. Documents in these results can be downloaded directly from the owner via a direct connection, or the client can perform another search with refined keywords.

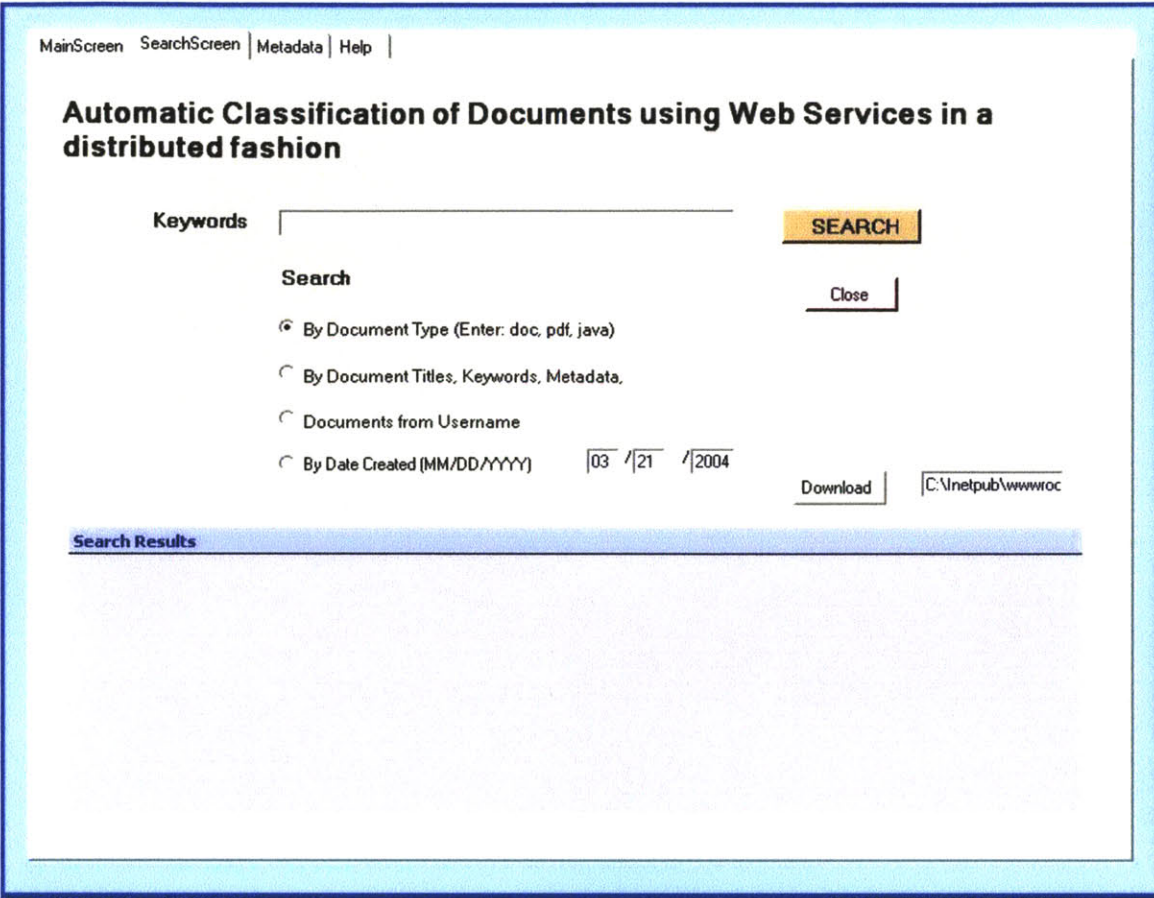


Figure 6.3.2- 2: I-Document Search Page

The help tab simply lists any problems or errors that a user may encounter while using the application. It also includes directions for necessary downloads and security modifications to make I-Document work proficiently. If a user cannot find an answer to a problem, he can contact one of the developers (information on the developers is available on the product Web site). Otherwise, restarting the application typically solves errors from a fully-functional application.

The metadata optimization tab (Figure 5.2- 1) is an interesting feature that works with Microsoft Office documents. With this functionality, users can view documents' metadata and modify that information as they see fit. This tab encourages clients to

update their metadata so other users are provided up-to-date information in a search. Currently, this metadata search is only set for Word documents. Another application (or tester program) was created that works with PowerPoint and Excel, but unknown problems were encountered with the code transfer. Thus, this functionality is kept simple for the first version of I-Document.

6.3.3 Database

The database (Figure 6.3.3- 1) stores user and document information. Currently, I-Document uses Microsoft Access and SQL Server because the programmers are familiar with these programs. Two tables on the database are used for client information. One table lists the clients and their IP addresses (used for P2P connections). The other user table stores usernames and password. A third table stores all document information. This data includes keywords, file paths, user names, authors, and metadata. Each user index is automatically reclassified and updated according to the chosen cycle time, and this index is eliminated from the server upon the client logging off from the network. This elimination saves space and keeps information on the server up-to-date. It also prevents users from trying to download files from clients not connected.

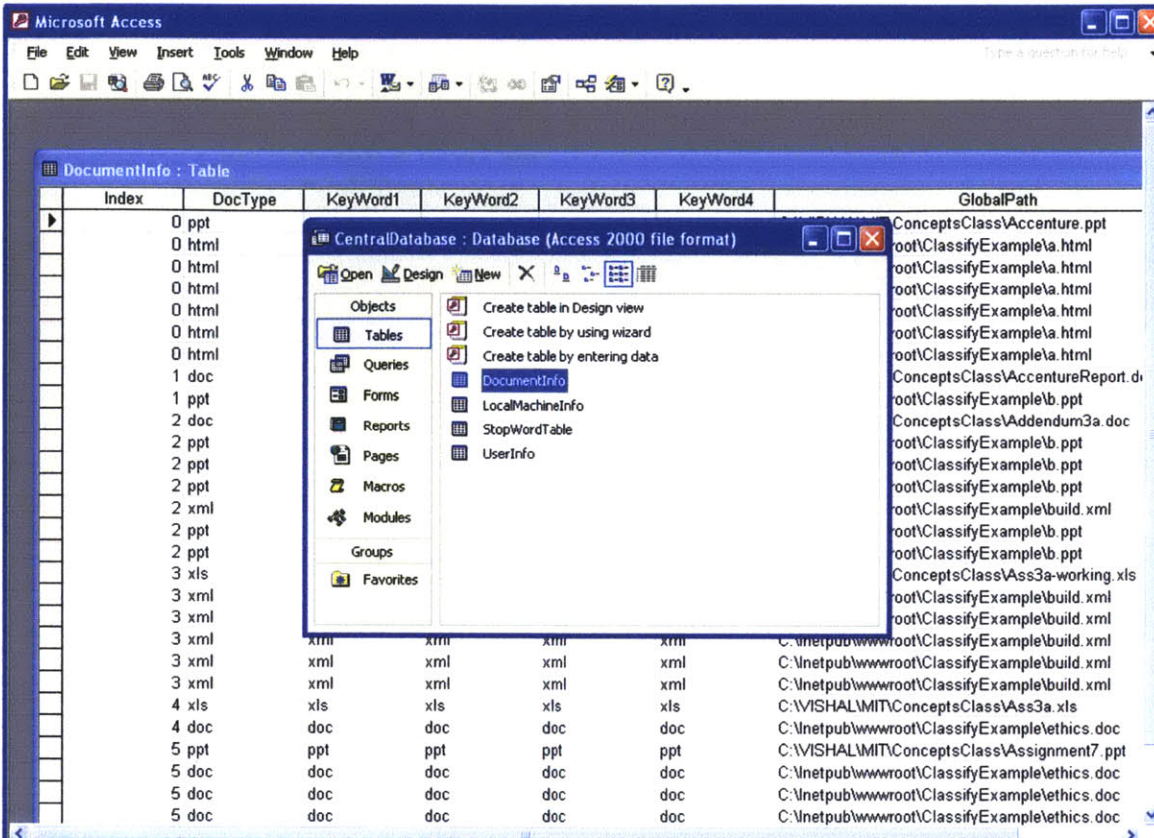


Figure 6.3.3- 1: I-Document Database

The final table is the stop list. A stop list is used to eliminate common words from being stored as keywords in text summarization operations. Thus, common words such as “won’t”, “very”, “to”, “an”, and “think” will not be stored in the document table unless they are stored as the metadata or the title. This database table is the main stop list that contains about eight hundred of the thousand most popular words in the English language. There is also a smaller list of about thirty words on the local service for the first stage of automatic classification. Thus, keywords are collected once and analyzed twice against increasingly specific stop lists. The location of the stop lists are separated so the larger list can be modified by a central authority. If a certain word becomes popular, it can be easily added to the stop list.

6.4 Assessment of the System

Significant testing of I-Document revealed solid results in regards to Information Retrieval and Information Extraction. Using a small test base of eight users and seventy documents, the system was analyzed in regard to how it would respond to searches through the grid. Tests were also conducted to evaluate the classification algorithm's extraction of relevant keywords.

6.4.1 Information Retrieval Assessment

Searches through the database extracted all relevant documents according to a user's query. If a search was by username, the system would return all documents stored by that specific user. Otherwise, if the search was by keywords, metadata, or date, and the searched term or date was in the document table, those corresponding documents would be returned. No variations were encountered. Small problems did arise as tests were conducted with plural versions of keywords or misspellings. The system would return nothing if, for instance, a client typed in "automobiles" and the only stored keyword was "automobile". Without a stemming or auto-correction algorithm, I-Document cannot generalize words or fix users' spelling mistakes. Thus, users must refine their searches and check the spelling of the words.

Returning all documents that had a keyword match somewhat questions the issue of relevance, but this is the operation inherent in a Boolean Information Retrieval System. Implementing a weighting algorithm was not possible because time was not available. There was also no desire from the designers to influence the returned list of documents.

But while the keywords may match in a search, the retrieved documents may not be what the user is seeking. This issue of relevance pertains to the precision ratio:

$$P = (\text{Relevant items retrieved}) / (\text{All items retrieved}) \quad (\text{Equation 4})$$

Essentially, this formula determines the utility of the search results. This ratio is very difficult to determine with I-Document because the test case involved a small document base. It is also challenging to assume what a user thinks or desires. Finally, I-Document was not coded for absolute precision. It was coded for recall.

The recall ratio measures the proportion of returned documents retrieved out of the document base. The formula is written as:

$$R = (\text{Relevant items retrieved}) / (\text{All items}) \quad (\text{Equation 5})$$

By this formula, I-Document is highly successful. Any match of keywords initiates retrieval, and the system did not pull any documents that lacked the query terms. By the Boolean model, any match leads to retrieval because the document is considered relevant if a query term pairs up with a word stored in the database. Unfortunately, this method does create a high degree of fallout. Fallout is defined as the proportion of documents that lack importance to the user conducting the search. By returning all matching documents, most will not pertain exactly to a user's objective.

Presenting the user with all corresponding documents, despite a low or high significance ratio, is not the optimal way of retrieval. This is especially true as the number of users

and documents grow. Therefore, future versions of I-Document will need some sort of ranking system to encourage a more precise list of retrieved documents, or at least a weighted list (more relevant documents returned at the top of the list).

6.4.2 Information Extraction Assessment

The algorithm for the document text extraction is basic, but it works according to the needs of the I-Document system. The algorithm extracts the four most frequent words not included in the stop list. These words may not exactly represent the meaning of the document, but in theory they should provide a significant clue. This operation had a 100% success rate with .txt files. The algorithm did, however, take a longer time to compile than originally expected. Classifying thirteen documents, of which three were txt documents, took, on average, one minute and twenty seconds to complete.

The other algorithms for metadata and file properties do exactly what they are supposed to do. They go into each file, extract significant properties, and store them in the database. With all of these classification methods working, the meaning of the document should come across in the database. Obviously, there is much room for improvement. Ideas for this improvement have already been mentioned in Section 4.4, but currently the system works according to the original design. It is supposed to begin with a framework. More efficient algorithms can always be added. Relevance is central to I-Document. It is central to competent communication. Thus, improvements are eventually necessary to narrow down searches.

7.1 I-Document Competition

Competition is a problem. Information Retrieval is one of the most important research topics in Computer Science. Thus, many companies, developers, and academics are searching for a forefront solution. While this software is unique with the adoption of Web services and P2P computing, it still faces heavy opposition. Four major areas of competition are considered: standard search engines, data repositories/digital libraries, small commercial systems, and other P2P applications.

7.2 Internet Search Engines

Many people refer to the World Wide Web as the largest library in the world. The amount of data on the Internet grows exponentially. With such growth, fast and intelligent tools are needed to search this vast resource, and search engines serve this purpose. Conventional Information Retrieval techniques utilized in Web search engines, however, are far from perfection in terms of optimal recall and precision searching.

There are many problems associated with popular search engines. First, measuring document change takes too long. The Internet is crawled by search engine tools (called spiders), and collections of documents and pages are gathered and stored locally. This crawling rate can take twenty to thirty days to refresh the collection since search engines like Google, AlltheWeb, and Inktomi have about three billion pages in their databases. Thus, fresh information on Web sites that are updated daily (hourly for some news sites) do not appear in search engine databases until almost a month after posting. One

suggested method of solving this problem is crawling Web sites that have a greater rate of change more frequently. The other suggestion is building a system to churn through about a trillion possible results in milliseconds (this second solution is far off in terms of technology).

Second, Web search engines' PageRank system has flaws. Google exploits links to measure the importance of Web sites. Thus, the more pages that are linked to a certain site, the more important that referenced site becomes. Researchers Sung Jin Kim and Sang Ho Lee recently discovered that the algorithm used to devise these rankings is not always optimized. There is a problem with computing several vector-based multiplications. There is also a problem with the concept. Programmers can take advantage of the system to make their site more popular, or, on the other hand, they can make a site popular under an unusual phrase (called a "Google Bomb"). In the first situation, spammers blog thousands of pages to win the attention of Google. If successful, the spammer's site will top the results list for a certain keyword or phrase. In the latter situation (Google Bomb), spammers with a sense of humor popularize a site under a single phrase. For an example, Internet users have attributed the phrase "miserable failure" with the URL www.whitehouse.gov/president/gwbbio.html. Finally, there is a problem associated with paid inclusion on the ranking system. Search engines like Yahoo!, Ask Jeeves, and Microsoft allow companies to increase their rank in the search results with their check books. Often, there are no distinguishing marks to alert search engine users of the company that paid to be included under certain keywords. Therefore, it is now questionable when using these sites if the results are accurate.

Third, classification schemes are imposed on the user. Thus, the user cannot decide if he or she wants to search through full text, metadata, or other information. For now, many search engines do not utilize other Web page information except full text and the link structure. I-Document allows users to choose their method of inquiry.

The difference between I-Document and Web search engines is extensive. First, searching and classification is pushed out to the client level. Thus, document update frequency is real-time. Second, I-Document does not implement a PageRank system. All results are displayed that match a keyword. Eventually, a ranking system will be imposed that will be based on the relevancy of the document according to the search. Monetary kickbacks will never be a part of this system. Third, I-Document combines multiple document representations to accurately classify documents. Users can choose which of those representations they wish to utilize.

7.3 Data Repositories/Digital Libraries

The use of data repositories and digital libraries is slowly growing. In this architecture, entire documents are all stored in a central location. This location offers services such as submission, searching, and retrieval. The collection of documents is organized and the entire document base is stored digitally. Some real world examples of such systems include DSpace, Phronesis, and small intra-company server systems. Of course, problems arise with large, server-based storage systems.

The first obvious problem is space. A large physical space is required to store a collection of documents. Such size brings with it issues regarding an expensive technical infrastructure. There is also a problem attributed to scalability. By nature, repositories are designed to grow and remain effective as they increase in volume. However, with greater usage, query times slow down and available space quickly decreases.

A second problem surrounds maintenance activities. Documents are placed in libraries and repositories and can remain there well beyond their age of relevance or use. Storage systems do not eliminate data unless directed to do so. Users can simply forget or not care to replace their older documents with up-to-date versions. This eliminates useful space and does not assist efforts toward useful data extraction.

A third issue is the proper management of intellectual property. Public and private repositories need some system in place to identify confidential, copyrighted, and published material. Within companies, there are security issues over private data. A common problem is discovering how to limit access to certain documents in the server. Restricting access to documents and securing digital rights will continue to be a problem unless free access to information is imposed.

A final problem concerns server failure. If the central storage system shuts down, or worse, is exposed to a virus and experiences data corruption, necessary information can be lost when it is needed the most. Major corporations lose thousands of dollars when their central server temporarily shuts down. Imagine the cost of losing the stored

information. Thus, backup systems, virus protection software, and reliable components are necessary for the success of the repositories. This simply equals more money to provide safety of mind.

The I-Document application has several competitive advantages over repositories. It first does not require a large physical space since all documents are stored and searched locally. Maintenance is not a problem with automatic updates. Intellectual property is currently not a concern because all shared information will be freely accessible for the academic version of this product. Central failure is the only matching problem with repositories. However, the only information backup required is for registered user information, and the Web service can be quickly transferred to another server. Finally, I-Document's technical infrastructure requirements are not extensive or expensive in comparison to digital libraries and data repositories.

7.4 Small Commercial Systems

Small commercial systems are hard to contrast against since there are many competitors in the text-mining market, and many of them take different directions in their solutions. Through extensive researching, four primary examples of information retrieval companies are gathered for this comparative section: Teragram, Temis, Cymfony, and conceptSearching.

These companies have similar visions and technologies. They all provide leading technologies and services for extracting key information from unstructured documents.

Thus, their customer base amounts to the same group: businesses, corporations, and governments. Teragram Corporation specializes in linguistic technologies along with search and retrieval. Temis' software solutions "enhance unstructured information analysis"⁶. Cymfony has powerful extraction tools and provides natural language processing, all at real time. They also tend to focus on the media customer segment. Finally, conceptSearching uses probability theory to extract and rank the relevance of information. These companies, and several like them, all provide information extraction solutions for those who can afford it.

I-Document, frankly, cannot compete in terms of technology. Private companies have greater resources and deeper pockets. I-Document is a framework that provides a free academic solution to automatic classification and information retrieval. While commercial systems have the assets to provide multiple and extensive solutions, academic attempts often only focus on one area of research. Money, expertise, and time are limited in academic attempts. Thus, corporate competition is a force that must be ignored for now.

7.5 Other P2P Applications

A great advantage with peer-to-peer computing lies in distributing the load on a given machine. Thus, files can be stored on different systems and shared, or individual client machines can be harnessed to run in parallel (SETI@home for example). With its overwhelming benefits looming to be tapped, many players are getting into this

⁶ <http://www.temis-group.com>

development game. The players can be split into two major groups: media focus and industry.

The applications focused around media make up a very long list. Examples include Napster, Kazaa, Morpheus, BearShare, LimeWire, and Gnutella. These efforts primarily concentrate on sharing digital music, videos, pictures, and software. They do not center on documents, and they do not incorporate a classification medium.

Two of the big industry players are Microsoft and Sun. Microsoft is developing Farsite, a “serverless, distributed file system... [where] computers collaboratively establish a virtual file server that can be accessed by any of the clients”⁷. Sun is working on a similar system called JXTA (Project Juxtapose). JXTA is a set of protocols centered on Java that establishes a virtual network between peers. Both Microsoft and Sun’s systems are in the preliminary stage, but once improved and stabilized, they should have a revolutionary effect on computing. Applications for these systems, along with the optimization of the frameworks, are in the planning and engineering phase.

With I-Documents text and classification focus, it stands out among the other developed P2P systems. I-Document is geared toward education and research, not music. I-Document also stands out with its .Net infrastructure, enabling the incorporation of multiple languages and platforms.

⁷ <http://research.microsoft.com/sn/Farsite/overview.htm>

8.1 Attacking the Market

Knowing the competition, it became much easier to analyze the market and write a basic strategy for attacking it. If this project group has the time and resources to release I-Document to the public, we would do so in a way to ensure the most success. The release would take into account the customer segmentation, competitive advantages, and driving business forces.

8.2 Customer Segmentation

The market base, in terms of I-Document, is split into three main groups: education, public, and corporate. I-Document's release would first begin with a free distribution to the academic bucket of users since this application is geared for academic and research documents. Thus, its use and popularity would be built at institutions like the Massachusetts Institute of Technology. M.I.T.'s Courseware initiative is an optimal place to launch since Courseware's use is worldwide. Students could share class documents with each other and external users in an effort to expand understanding of the material. This will also work well in fostering an electronic mentor environment.

Once I-Document is established at M.I.T., its functionality would be extended before pitching it to other universities. To gain revenue, advertisements can be added to the Web site and the product, but I-Document will always remain free in any academic setting. If money is needed for development, the order of financing options would move from self-financing to angel funding and lastly venture capitalism. From universities, I-

Document will naturally gain public scrutiny. A public release strategy may not be necessary if it has enough academic fanfare.

With the success built from the educational and public markets, a focus would then be placed on the corporate segment and the money that comes with it. There are several e-document intensive businesses (law firms, consulting companies, manufacturers, etc.), and many of them have an interest in relevant data extraction. Essentially, I-Document's business strategy would shift from free public access to private, customizable access. The company would become a software creator, a consultant, and a service provider. Reengineered and enhanced versions of I-Document would need to be created to offer a secure, portable solution that works real time in company networks and beyond firewalls (for the traveling employee). If this market segment is reached, the road would get bumpy, but the profits would be tremendous.

8.3 Competitive Advantages

There are three main factors to acquire and maintain a competitive advantage: technology, differentiation, and cost. Technology must stay ahead of the curve. I-Document's business would revolve around nonstop research and the integration of proven theories. Code to interconnect new devices and operating languages would also need to be infused to extend the application's reach. This is the whole point of distributed computing. I-Document needs to work successfully across multiple platforms to be universally accepted and favored.

The degree of differentiation must be above the norm to maintain specialty. There is already an existing need for I-Document, but it needs to stand above the crowd of competitors. This is achieved through a solid product, popularity, and a service niche. Automatic classification through Web services and P2P computing is the niche.

Finally, pricing must be competitive. At first, the pricing is on target with a zero dollar cost. This builds a user base. For the corporate deployment, the price tag will go up significantly, but it should not go beyond the competitors' prices. Below is a chart displaying the competitive advantage by looking at differentiation and relative costs. I-Document is represented by the circle.

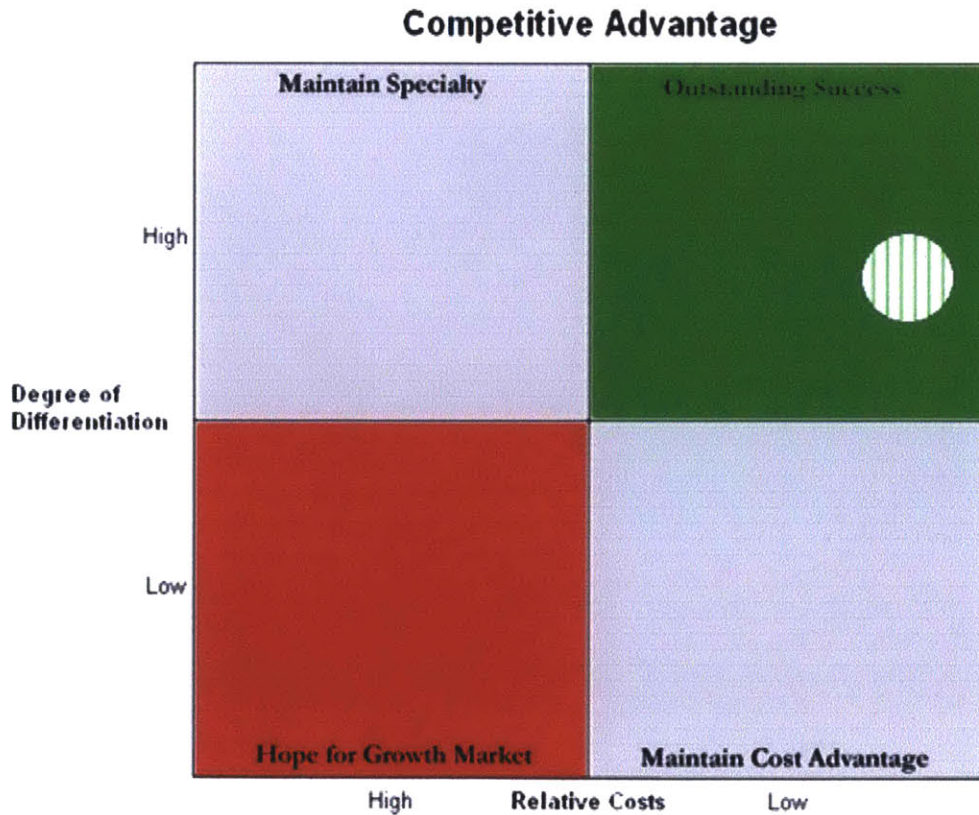


Figure 8.3- 1: I-Document Competitive Advantage Chart

8.4 Business Forces

The driving forces behind I-Document can be summarized by the modified 5-forces model (Figure 8.4- 1). Surrounded by the strategic business unit, there are five main considerations for a business venture. First, the possibility of new entrants is high since the cost to enter the automatic classification/information retrieval market is low. In addition, there are many existing competitors. Thus, the I-Document management must focus on the strategies explained in the competitive advantages section.

Second, the I-Document team must worry about the customer. It must lock in education institutions and then work on corporate customers. I-Document only entails a localized exploitation in business transformation. This means that it will cause a minimal process change of operations. Customers must see that it is a minor change to their computer networks, but at the same time, it has enormous potential. It will enhance their operations and business practices.

The third area of concern is the supplier. There is currently one main supplier – Microsoft. While this is not usually a good aspect of this force, Microsoft is not expected to break up any time soon or dramatically increase its prices. Thus, this segment is not a concern.

The fourth force, which is not represented in Figure 8.4- 1 since the figure is the modified diagram, represents the degree of rivalry. Firms fight for a competitive advantage over other companies. This rivalry depends on the concentration and the cutthroat nature of

the industry. Regarding concentration, there are a couple large firms and several small firms in Information Extraction. Despite these few large firms, this industry does not have a monopolistic environment. The total number of firms is also not overwhelming – decreasing the concentration level. Thus, the concentration level is at a medium rating. There is also a low level of intensity in terms of pricing. Overall, the strength of rivalry is not high in the information retrieval market. It is a mid-level force.

The fifth and final force is new products and services. This links with the customer force and demonstrates the benefits I-Document will deliver. Using new technologies (Web services, P2P, and .Net), I-Document enhances the operation of business and relevant information retrieval. These technological choices, however, are not completely strategic. They are publicly available, and private companies are gaining rapid interest in their benefits and use.

5-Forces

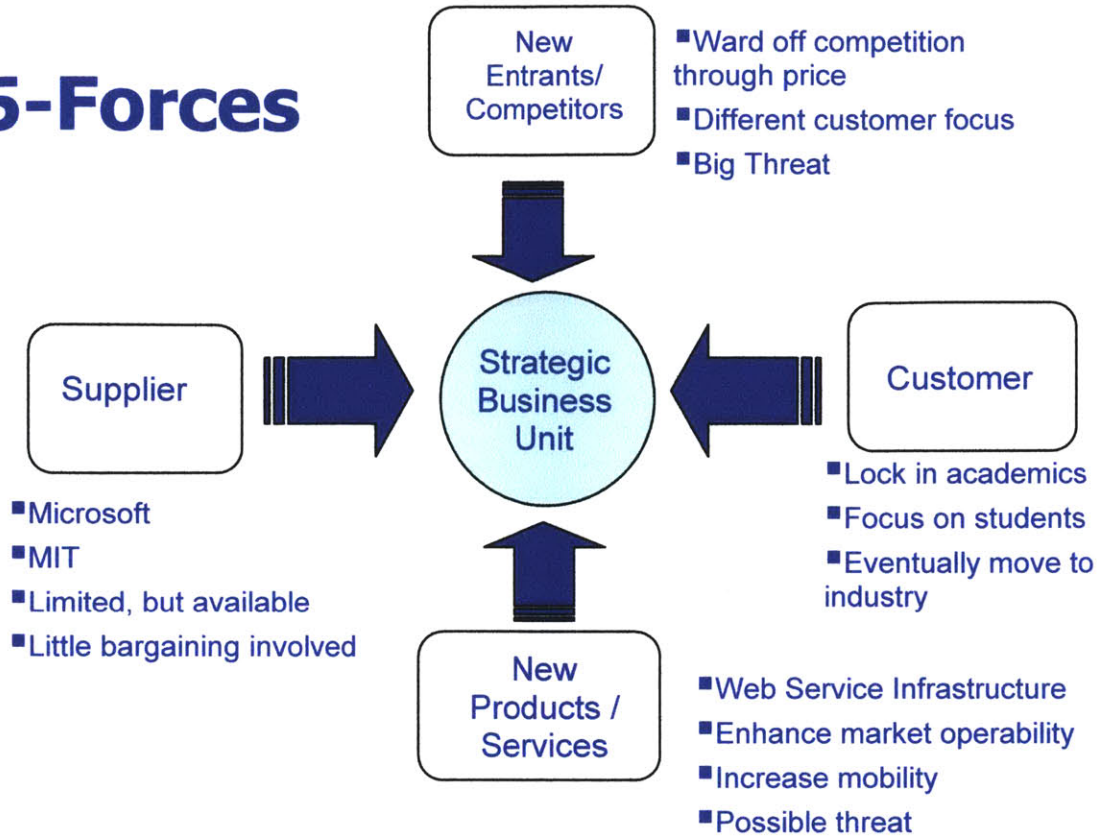


Figure 8.4- 1: 5-Forces Model for I-Document

8.5 Competition Conclusion

Through the I-Document team's focus on customer segmentation, competitive advantages, and driving forces, the company will be initially prepared for transferring I-Document into a public venture. This release is unlikely, however, since the original team is likely to split up after graduation. We have all taken jobs in different industries across the country. Thus, another group would need to take on this endeavor.

9.1 Thesis Conclusion

The main objective of this thesis project was to code a fully operational Web application that automatically classifies documents utilizing scalable, distributive computing, share this classification across a grid of users, and provide a method for searching through and downloading these documents via a peer-to-peer connection. I-Document is the result of this goal. In the process of programming I-Document, which took a full three months, investigation of current Information Retrieval, Information Extraction, and text summarization techniques was also required to create a competitive and advanced application. While a productized version of this vision was not attempted, I-Document, nonetheless, provides a solid framework for classifying and freely sharing digital documents.

References

- Abramowicz, Witol. Knowledge-Based Information Retrieval and Filtering from the Web. Boston, U.S.A.: Kluwer Academic Publishers, 2003
- BCS-IRSG European Colloquium on IR Research (24th : March 25-27, 2002 : Glasgow, Scotland). Advances in Information Retrieval. Fabio Crestani (ed.), Mark Girolami (ed.), Cornelis Joost van Rijsbergen (ed.). Berlin, Germany: Springer-Verlag, 2002
- Brain, Marshall. "How Boolean Logic Works". HowStuffWorks, Inc. <<http://computer.howstuffworks.com/boolean.htm>>. 2004
- Bray, Hiawatha. "Paying for Listing with Search Engines". Boston Globe, Page C3. March 8, 2004
- Center for Intelligent Information Retrieval. "NSF Digital Government Project". CIIR. <<http://ciir.cs.umass.edu/projects/>>. 2004
- CICLing 2003 (2003 : Mexico City, Mexico). Computational Linguistics and Intelligent Text Processing : 4th International Conference, CICLing February 16-22, 2003. Alexander Gelbukh (ed.). Berlin ; New York: Springer, 2003
- conceptSearching. "Information Retrieval (IR) system based on the Probabilistic Model". conceptSearching. <<http://www.conceptsearching.com/conceptFAQ.htm#a2>>, 2004
- Cymfony. Main Web Site. Cymfony, Inc. <<http://www.cymfony.com/>>. 2004
- Dominich, Sandor. Mathematical Foundations of Information Retrieval. Dordrecht ; Boston: Kluwer Academic Publishers, 2001
- DSpace. "MIT's digital repository". MIT and Hewlett-Packard. <<https://dspace.mit.edu/index.jsp>>. 2002
- Dublin Core Metadata Initiative. "History of the Dublin Core Metadata Initiative". DCMI and OCLC Research. <<http://dublincore.org/about/history/>>. 2004
- Engenium. "Engenium Semetric Overview". Engenium. <<http://www.engenium.com>>, 2004
- Farsite. Microsoft Corporation. <<http://research.microsoft.com/sn/Farsite/overview.htm>>. 2004
- Google. "The Basics of Google Search". Google. <<http://www.google.com/help/basics.html#stopwords>>. 2004

- Grehan, Mike. "Google PageRank Lunacy". Search Engine Watch. <<http://searchenginewatch.com/searchday/article.php/3319461>>. March 4, 2004
- Hagel III, John and Brown, John. "Your Next IT Strategy". Harvard Business Review, Pages 105-112. Harvard Business School Publishing Corporation: October 2001
- Johnson, Steven. "The (Evil) Genius of Comment Spammers" – Googlemania. Wired Magazine, Page 119. March 2004
- Jones, Jerry. "Vocabulary Workshop: 1000 Most Common Words in English". About, Inc. <http://esl.about.com/library/vocabulary/bl1000_list1.htm?PM=ss14_esl>. 2004
- Jones, Susan. Text and Context : Document Storage and Processing. London ; New York: Springer-Verlag, 1991
- Joyce, James. "Bayes' Theorem". *The Stanford Encyclopedia of Philosophy (Winter 2003 Edition)*. Edward N. Zalta (ed.). <<http://plato.stanford.edu/archives/win2003/entries/bayes-theorem>>. 2003
- JXTA. "JXTA, Sun Microsystems' project". PC Magazine. <<http://pcmag.dit.net/article.php?id=EpEuZEplyVFwMWqUIP>>. Tuesday, February 26, 2002
- JXTA. "Project JXTA". Sun Microsystems, Inc. <<http://www.jxta.org/>>. 2003
- Malone, Michael S. "Surviving IPO Fever" – Googlemania. Wired Magazine, Page 116. March 2004
- Mangalindan, Mylene. "Yahoo Search Results to Include Paid Links". The Wall Street Journal, Page D1. Tuesday, March 2, 2004
- McConnell, Steve. Rapid Development. Redmond, Washington, USA: Microsoft Press, 1996
- McHugh, Josh. "It's an Ad, Ad, Ad, Ad World" – Googlemania. Wired Magazine, Page 123. March 2004
- McKay, Bain. "Leveraging corporate knowledge through automatic classification". Zatz Publishing. <<http://www.dominopower.com/issues/issue200002/autoclass001.html>>, 2004
- Microsoft Corporation. "HOWTO: User Automation to Get and to Set Office Document Properties with Visual C# .NET". Microsoft Corporation. <<http://support.microsoft.com/?kbid=303296>>. December 15, 2003

Microsoft Corporation. "INFO: Microsoft Office XP PIAs Are Available For Download". Microsoft Corporation.
<<http://support.microsoft.com/default.aspx?scid=kb;EN-US;328912>>. December 15, 2003

Phronesis. "Phronesis Project". Tecnologico de Monterrey, Campus Monterrey.
<<http://copernico.mty.itesm.mx/phronesis/project/Default.html>>. 2004

Porter, Martin. "The Porter Stemming Algorithm".
<<http://www.tartarus.org/~martin/PorterStemmer/>>. 2004

SOAP. Webopedia's definition of SOAP. Jupitermedia Corporation.
<<http://sbc.webopedia.com/TERM/S/SOAP.html>>. 2004

South American Symposium on String Processing and Information Retrieval (10th : October 8-10, 2003 : Manaus, Brazil). String Processing and Information Retrieval. Mario A. Nascimento (ed.), Edleno S. de Moura (ed.), Arlindo L. Oliveira(ed.). Berlin ; New York: Springer-Verlag, 2003

Sridharan, Prashant. "Microsoft Programming Languages". Microsoft Corporation.
<<http://msdn.microsoft.com/vstudio/productinfo/whitepapers/default.aspx>>. July 2003

Temis. "Text Mining Solutions. Temis-Group. <<http://www.temis-group.com>>. 2004

Teragram. "Practical Solutions to Monstrous Amounts of Information". Teragram Corporation. <<http://www.teragram.com>>. 2004

WordNet 2.0. "WordNet: a lexical database for the English language". Princeton University. <<http://www.cogsci.princeton.edu/~wn/>>. 2004

Wrox Press. "Metadata". Developer Fusion Ltd.
<<http://www.developerfusion.com/show/1678/3/>>. 2003

Appendix 1: Metadata Code

The following code is used in the Web application of I-Document. It governs the Metadata tab and extracts useful Metadata information. This code (slightly modified) is also used in the Web service for classification means.

ADDED REFERENCES INCLUDE:

```
adodb  
Microsoft.Office.Core  
msdatasrc  
office  
stdole  
VBIDE  
Word
```

ADDED SYSTEM FILES FOR METADATA TO WORK:

```
using Microsoft.Office.Core;  
using System.Reflection;  
using Word = Microsoft.Office.Interop.Word;  
using System.Runtime.InteropServices;
```

APPLICATION LEVEL VARIABLES:

```
public class ClientInterface : System.Windows.Forms.Form  
  
    //Metadata added March 14  
    private Word.Document aDoc;  
    private string extension;  
    private object fileName;  
    private object aDocBuiltInProps;  
    private Type typeDocBuiltInProps;  
    private Word.ApplicationClass WordApp;  
    private System.Windows.Forms.Button button4;  
    private System.Windows.Forms.ListBox listBox1;  
    private System.Windows.Forms.Button button5;  
    private System.Windows.Forms.Label label16;  
    private System.Windows.Forms.ComboBox comboBox1;  
    private System.Windows.Forms.Label label17;  
    private System.Windows.Forms.TextBox textBox2;  
    private System.Windows.Forms.Button button6;  
    private System.Windows.Forms.Label label10;  
    private System.Windows.Forms.Button CloseButton1;
```

CODE FOR THE METADATA TAB:

```
//Reference: http://support.microsoft.com/?kbid=303296  
//Brandon Hohm, March 14  
private void button4_Click(object sender, System.EventArgs e)  
{  
    listBox1.Items.Clear();  
    listBox1.Items.Add("Please choose a Word file");  
  
    try  
    {  
        if (this.openFileDialog1.ShowDialog() == DialogResult.OK)  
        {  
            //Gets file name, if doc, executes  
            //if not, sends it to the right function  
            fileName = openFileDialog1.FileName;
```

```

GetExtension(fileName);

if(extension == ".doc")
{
    object readOnly = false;
    object isVisible = false;
    object missing = System.Reflection.Missing.Value;
    WordApp = new Word.ApplicationClass();
    WordApp.Visible = false;
    aDoc = WordApp.Documents.Open(ref fileName, ref
        missing,ref readOnly, ref missing, ref missing,
        ref missing, ref missing, ref missing, ref
        missing, ref missing, ref missing, ref
        isVisible,ref missing,ref missing,ref missing);

    //aDoc.Activate();
    aDocBuiltInProps = aDoc.BuiltInDocumentProperties;
    typeDocBuiltInProps = aDocBuiltInProps.GetType();

    //Get the Title property and display it.
    string strIndex = "Title";
    string strValue;
    object aDocTitleProp =
        typeDocBuiltInProps.InvokeMember("Item",
            BindingFlags.Default |BindingFlags.GetProperty,
            null,aDocBuiltInProps, new object[] {strIndex});
    Type typeDocTitleProp = aDocTitleProp.GetType();
    strValue = typeDocTitleProp.InvokeMember("Value",
        BindingFlags.Default |BindingFlags.GetProperty,
        null,aDocTitleProp, new object[] {} ).ToString();
    listBox1.Items.Add( "The Title is: " + strValue);

    //Get the Subject property and display it.
    string strIndex2 = "Subject";
    string strValue2;
    object aDocSubjectProp =
        typeDocBuiltInProps.InvokeMember("Item",
            BindingFlags.Default |BindingFlags.GetProperty,
            null,aDocBuiltInProps,new object[] {strIndex2});
    Type typeDocSubjectProp = aDocSubjectProp.GetType();
    strValue2 = typeDocSubjectProp.InvokeMember("Value",
        BindingFlags.Default |BindingFlags.GetProperty,
        null,aDocSubjectProp,
        new object[] { } ).ToString();
    listBox1.Items.Add( "The Subject is: " + strValue2);

    //Get the Author property and display it.
    string strIndex3 = "Author";
    string strValue3;
    object aDocAuthorProp =
        typeDocBuiltInProps.InvokeMember("Item",
            BindingFlags.Default |BindingFlags.GetProperty,
            null,aDocBuiltInProps,
            new object[] {strIndex3} );
    Type typeDocAuthorProp = aDocAuthorProp.GetType();
    strValue3 = typeDocAuthorProp.InvokeMember("Value",
        BindingFlags.Default |BindingFlags.GetProperty,

```

```

        null,aDocAuthorProp,new object[]{}).ToString();
listBox1.Items.Add( "The Author is: " + strValue3);

//Get the Company property and display it.
string strIndex4 = "Company";
string strValue4;
object aDocCompanyProp =
    typeDocBuiltInProps.InvokeMember("Item",
        BindingFlags.Default |BindingFlags.GetProperty,
        null,aDocBuiltInProps,
        new object[] {strIndex4} );
Type typeDocCompanyProp = aDocCompanyProp.GetType();
strValue4 = typeDocCompanyProp.InvokeMember("Value",
    BindingFlags.Default |
    BindingFlags.GetProperty,
    null,aDocCompanyProp, new
    object[]{}).ToString();
listBox1.Items.Add( "The Company is: " + strValue4);

//Get the Keywords property and display it.
string strIndex5 = "Keywords";
string strValue5;
object aDocKeyProp =
    typeDocBuiltInProps.InvokeMember("Item",
        BindingFlags.Default |BindingFlags.GetProperty,
        null,aDocBuiltInProps,
        new object[] {strIndex5} );
Type typeDocKeyProp = aDocKeyProp.GetType();
strValue5 = typeDocKeyProp.InvokeMember("Value",
    BindingFlags.Default |
    BindingFlags.GetProperty, null,aDocKeyProp,
    new object[] { } ).ToString();
listBox1.Items.Add("The Keywords are: " + strValue5);

//Get the Comments property and display it.
string strIndex6 = "Comments";
string strValue6;
object aDocCommProp =
    typeDocBuiltInProps.InvokeMember("Item",
        BindingFlags.Default |
        BindingFlags.GetProperty,null,aDocBuiltInProps,
        new object[] {strIndex6} );
Type typeDocCommProp = aDocCommProp.GetType();
strValue6 = typeDocCommProp.InvokeMember("Value",
    BindingFlags.Default |
    BindingFlags.GetProperty,null,aDocCommProp,
    new object[] { } ).ToString();
listBox1.Items.Add("The Comments are: " + strValue6);
}

else
{
    string message = "ERROR: Please choose an Office
    Word file" ;
    MessageBox.Show(message);
}
}

```



```

    }
    catch
    {
        aDoc = null;
        fileName = null;
        extension = null;
        MessageBox.Show("Problem with opening document. Please try
            again.");
    }
}

```

```

private void openFileDialog1_FileOk(object sender,
System.ComponentModel.CancelEventArgs e)
{
}

```

//Save and close document - March 14

```

private void button5_Click(object sender, System.EventArgs e)
{
    try
    {
        object ignore = null;
        object saveChanges = true;
        aDoc.Save();
        listBox1.Items.Add("saved");
        aDoc.Close(ref saveChanges, ref ignore, ref ignore);
        aDoc = null;
        fileName = null;
        extension = null;
        //Need to quit WordApp or else the application keeps
            running behind the scenes
        WordApp.Quit(ref saveChanges, ref ignore, ref ignore);
        // Marshal.ReleaseComObject(WordApp);
    }

    catch(Exception ex)
    {
        Console.WriteLine(ex);
        string message = "ERROR: Please stop pushing the save
            button when the doc is closed" ;
        MessageBox.Show(message);
    }
}

```

//Changes to metadata - march 14

```

private void button6_Click(object sender, System.EventArgs e)
{
    try
    {
        if (comboBox1.SelectedItem != null && textBox2.Text != null)
        {
            if(extension == ".doc")

```

```

        {
            string strIndexNew = (string) comboBox1.SelectedItem;
            string strValueNew = textBox2.Text;

            object aDocInputProp =
                typeDocBuiltInProps.InvokeMember("Item",
                    BindingFlags.Default |
                    BindingFlags.GetProperty,
                    null, aDocBuiltInProps,
                    new object[] {strIndexNew} );
            Type typeDocInputProp = aDocInputProp.GetType();

            typeDocInputProp.InvokeMember("Item",
                BindingFlags.Default |
                BindingFlags.SetProperty,
                null, aDocBuiltInProps,
                new object[] {strIndexNew, strValueNew} );
            listBox1.Items.Add("New metadata is: " + strIndexNew
                + " : " + strValueNew);
            aDoc.Save();
        }
    }
else
{
    listBox1.Items.Add("PLEASE PICK A PROPERTY AND FILL IN TEXT");
}

catch
{
    string message = "ERROR: Please stop pushing the metadata
        button" ;
    MessageBox.Show(message);
}
}

```

```

//Gets the extensions and initial file info - March 14
private void GetExtension(object fileName)
{
    //Works for all files
    string name = (string) fileName;
    FileInfo info = new FileInfo(name);
    //Get name
    string name2 = info.Name;
    //Get size
    long ByteSize = info.Length;
    //build extension
    int ExtPosition = info.Name.IndexOf(".");
    int LengthOfFileName = info.Name.Length;
    extension = Path.GetExtension(name);
    //extension = info.Name.Substring(ExtPosition+1, (LengthOfFileName-
        ExtPosition)-1);
    //This extension method will not work with fileNames with more than one
        period
    //use FILE to get the date information about the file

    DateTime CreatedOn = File.GetCreationTime(name);
}

```

```

        DateTime LastModified = File.GetLastWriteTime(name);

        listBox1.Items.Add("<<<<<<File Information>>>>>>");
        listBox1.Items.Add("Name: " + name2 + "          Size: " + ByteSize);
        listBox1.Items.Add("Created on: " + CreatedOn + "          Last
                            Modified: " + LastModified);

        //Here, use extension to send it to another function for excel
        and powerpoint

    }

    //References
    /*
    http://support.microsoft.com/?kbid=303296
    http://support.microsoft.com/default.aspx?scid=kb;EN-US;303718
    http://www.c-sharpcorner.com/Code/2002/Mar/WordFromDotNet.asp
    */

```

NOTE: More extensive code was developed to work for Excel, txt, and PowerPoint documents, but it is not included in this draft because it mimics the code in shown here. There are only minor changes in the references, added system files, opening, and saving documents. For example, to handle Excel Workbooks, you would add the following:

ADDED REFERENCE FILES:

Excel
Graph

ADDED SYSTEM FILES:

Using Excel = Microsoft.Office.Interop.Excel;
Using Graph = Microsoft.Office.Interop.Graph;

ADDED APPLICATION LEVEL VARIABLES:

```

private Excel.Workbook eDoc;
private object eDocBuiltInProps;
private Type typeExcelBuiltInProps;
private Excel.ApplicationClass ExcelApp = new
    Excel.ApplicationClass();

```

CODE BASE:

```

//Changes metadata function
private void button5_Click(object sender, System.EventArgs e)
{

```

```

if (comboBox1.SelectedItem != null)
{
    if(extension == "xls")
    {
        string strIndexNew = (string) comboBox1.SelectedItem;
        string strValueNew = textBox1.Text;

        object eDocInputProp =
            typeExcelBuiltInProps.InvokeMember("Item",
                BindingFlags.Default | BindingFlags.GetProperty,
                null,eDocBuiltInProps, new object[] {strIndexNew} );
        Type typeExcelInputProp = eDocInputProp.GetType();
        typeExcelInputProp.InvokeMember("Item",
            BindingFlags.Default | BindingFlags.SetProperty,
            null,eDocBuiltInProps, new object[]
                {strIndexNew,strValueNew} );
        listBox1.Items.Add("New metadata is: " + strIndexNew + " :
            " + strValueNew);
        eDoc.Save();
    }
}

```

>>>>>CODE FOR CHANGES IS CONTINUED ACCORDING TO THE DOCUMENT TYPE<<<<<

```

//Excel stop function
private void stopnow()
{
    try
    {
        object ignore = null;
        object saveChanges = true;
        eDoc.Save();
        eDoc.Close(saveChanges, fileName, ignore);
        eDoc = null;
        fileName = null;
        extension = null;
    }
    catch(Exception ex)
    {
        string message = "ERROR: Problem saving Excel Document" ;
        MessageBox.Show(message);
    }
}

```

```

//Gets Excel metadata information
private void ExcelWork(object fileName)
{
    object readOnly = false;
    object isVisible = true;
    object missing = System.Reflection.Missing.Value;

    ExcelApp.Visible = true;
}

```

```

//Might have a problem with file name with 2nd file
eDoc = ExcelApp.Workbooks.Open((string) fileName, missing,
    readOnly, missing, missing, missing, missing, missing,
    missing, missing, missing, isVisible, missing, missing,
    missing);

eDoc.Activate();

eDocBuiltinProps = eDoc.BuiltinDocumentProperties;
typeExcelBuiltinProps = eDocBuiltinProps.GetType();

//Get the Title property and display it.
string strIndexe = "Title";
string strValuee;
object eDocTitleProp = typeExcelBuiltinProps.InvokeMember("Item",
    BindingFlags.Default | BindingFlags.GetProperty,
    null, eDocBuiltinProps, new object[] {strIndexe} );
Type typeDocTitleProp = eDocTitleProp.GetType();
strValuee = typeDocTitleProp.InvokeMember("Value",
    BindingFlags.Default | BindingFlags.GetProperty,
    null, eDocTitleProp, new object[] { } ).ToString();
listBox1.Items.Add( "The Title is: " + strValuee);

```

**>>>>>>>THE OTHER PROPERTIES ARE OBTAINED IN THE SAME
FASHION<<<<<<<<**

Appendix 2: Stop List

The following list is used by the text summarization function to extract and eliminate common English types from becoming keywords.

the	do	live	large	might	main	step	able
of	their	where	add	saw	enough	early	pound
to	if	after	even	far	plain	hold	done
and	will	back	land	sea	girl	west	drive
a	way	little	here	draw	usual	ground	stood
in	about	only	must	left	young	interest	contain
is	many	man	big	late	ready	reach	front
it	then	year	high	run	above	fast	teach
you	them	came	such	don't	ever	verb	week
that	would	show	follow	while	red	listen	final
he	like	every	act	press	list	six	gave
was	so	good	why	close	though	less	green
for	these	me	ask	real	feel	morning	oh
on	her	give	men	life	talk	ten	quick
are	long	our	change	few	soon	simple	develop
with	make	under	went	north	body	several	warm
as	thing	name	kind	open	dog	toward	free
I	see	very	off	seem	direct	war	minute
his	him	through	need	together	pose	lay	special
they	two	just	try	next	leave	against	mind
be	has	great	us	white	song	pattern	behind
at	look	think	again	begin	measure	slow	clear
one	more	say	point	got	door	center	tail
have	day	help	near	walk	product	person	produce
this	could	low	self	example	short	serve	fact
from	go	line	stand	ease	class	appear	inch
or	come	differ	own	group	question	map	multiply
had	did	turn	page	always	happen	rain	nothing
by	no	cause	should	those	complete	rule	course
word	most	much	found	both	area	govern	stay
but	my	before	answer	mark	half	pull	full
what	over	move	grow	often	order	cold	force
some	know	right	study	letter	south	notice	blue
we	than	boy	still	until	problem	unit	object
can	call	old	learn	care	piece	fine	decide

out	first	too	plant	second	told	certain	surface
other	who	same	cover	carry	knew	fly	deep
were	may	tell	sun	took	pass	fall	foot
all	down	does	four	eat	since	lead	system
there	side	set	between	room	top	cry	busy
when	been	three	state	began	whole	dark	test
up	now	want	keep	idea	heard	note	common
use	find	air	eye	stop	best	wait	possible
your	any	well	never	once	hour	plan	stead
how	new	also	last	base	better	figure	dry
said	part	play	let	hear	true	box	wonder
an	take	small	thought	cut	during	noun	laugh
each	get	end	cross	sure	hundred	field	thousand
she	place	put	hard	color	five	rest	ago
which	made	read	start	face	remember	correct	ran
check	ride	third	poor	collect	depend	throw	master
shape	cell	shall	lot	save	rub	shine	track
equate	believe	held	bottom	control	tube	property	shore
hot	fraction	describe	key	decimal	famous	column	division
miss	sit	cook	single	gentle	fear	select	sheet
brought	race	floor	stick	woman	sight	wrong	substance
tire	store	either	flat	practice	thin	gray	favor
bring	train	result	twenty	separate	hurry	repeat	connect
yes	sleep	burn	skin	difficult	colony	require	post
distant	prove	hill	smile	please	clock	broad	spend
fill	lone	safe	crease	protect	mine	prepare	chord
east	leg	cat	hole	noon	tie	salt	fat
among	wall	consider	trip	whose	enter	nose	glad
grand	catch	type	receive	locate	major	plural	original
ball	mount	law	row	ring	fresh	anger	share
yet	wish	bit	mouth	caught	search	claim	station
drop	sky	copy	exact	period	send	skill	dad
am	board	phrase	symbol	indicate	allow	women	bread
present	joy	tall	die	spoke	print	solution	charge
heavy	sat	roll	least	history	dead	silver	proper
position	written	finger	trouble	effect	spot	thank	bar
arm	wild	value	shout	electric	suit	match	offer
wide	instrument	fight	except	expect	current	especially	segment
sail	kept	lie	wrote	crop	lift	afraid	instant
material	grass	beat	seed	modern	continue	huge	populate
size	job	excite	tone	element	block	discuss	chick

vary	edge	view	join	hit	chart	forward	dear
settle	sign	sense	suggest	corner	hat	similar	reply
spea	visit	ear	clean	supply	sell	experienc	drink
general	past	else	break	bone	subtract	score	occur
ice	soft	quite	lady	rail	event	bought	support
matter	fun	broke	yard	imagine	particular	led	speech
pair	bright	case	rise	provide	deal	pitch	nature
include	gas	middle	bad	agree	term	mass	range
divide	month	kill	blow	thus	opposite	card	motion
felt	million	son	touch	won't	shoulder	band	path
perhaps	finish	moment	grew	chair	spread	rope	log
pick	happy	scale	mix	danger	arrange	slip	meant
sudden	hope	loud	cost	thick	camp	win	quotient
count	gone	speed	lost	guess	invent	evening	
reason	jump	method	brown	necessary	born	condition	
length	eight	pay	wear	sharp	determine	feed	
represent	meet	age	equal	create	quart	tool	
art	root	section	sent	wash	nine	total	
subject	buy	surprise	choose	bat	noise	basic	
region	raise	quiet	fell	rather	level	smell	
energy	solve	tiny	fit	crowd	chance	nor	
probable	whether	climb	flow	compare	gather	double	
bed	push	cool	fair	string	shop	seat	
egg	seven	design	bank	bell	stretch	arrive	