SEMANTIC SOLUTION TRACKING SYSTEM

eShelf – An Implementation of Semantic Search Engine

A PROJECT REPORT

Submitted by

CHANDIRASEKAR.T(05C21)
KARTHIKEYAN.N.G(05C42)
KRISHNAPRIYA.S(05C47)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

THIAGARAJAR COLLEGE OF ENGINEERING, MADURAI

(An autonomous institution affiliated to Anna University)

(An ISO 9001:2000 Certified Institution)

APRIL 2009

SEMANTIC SOLUTION TRACKING SYSTEM

eShelf – An Implementation of Semantic Search Engine

A PROJECT REPORT

Submitted by

CHANDIRASEKAR.T(05C21)
KARTHIKEYAN.N.G(05C42)
KRISHNAPRIYA.S(05C47)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

THIAGARAJAR COLLEGE OF ENGINEERING, MADURAI

(An autonomous institution affiliated to Anna University)

(An ISO 9001:2000 Certified Institution)

APRIL 2009

THIAGARAJAR COLLEGE OF ENGINEERING, MADURAI

(An autonomous institution affiliated to Anna University)

(An ISO 9001:2000 Certified Institution)

BONAFIDE CERTIFICATE

Certified that this project report **"Semantic Solution Tracking System, eShelf – An Implementation of Semantic Search Engine" is** a bonafide work of **T.Chandirasekar, N.G.Karthikeyan, and S.KrishnaPriya** who carried out the project work under my supervision.

SIGNATURE

Dr.S.Mercy Shalinie, Ph.D
Head of the Department
Department of CSE
Thiagarajar College of Engineering
Madurai-15

SIGNATURE OF THE GUIDE

Mrs.Tamilselvi.D,
Lecturer
Department of CSE
Thiagarajar College of Engineering
Madurai-15

Station: Madurai

Date:

Submitted for the "VIVA-VOCE" Examination held at Thiagarajar College of Engineering on _____.

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our sincere gratitude to our Honorable Correspondent **Mr.Karumuthu T. Kannan** for providing us with an opportunity and facilities to complete our project.

We wish to express my deep sense of gratitude to **Dr.ABHAIKUMAR.V**, Principal of Thiagarajar College of Engineering for his support and encouragement throughout this project work.

Also we express our sincere thanks to **Dr.MERCY SHALINIE.S,** Head of the Department of Computer Science and Engineering for her ardent guidance and supportive work for our project.

We owe our special thanks and gratitude to **Mrs.D.TAMILSELVI**, Lecturer, Department of Computer Science and Engineering for her valuable suggestions, involvement and support during the course of our project.

We are also indebted to all the teaching and non-teaching staff members of our College for helping us directly or indirectly by all means throughout the course of our study and project work.

Finally we take this opportunity to thank our **PARENTS** for their moral support and help. We also extend thanks to our **FRIENDS** who have been helpful for sharing their novel ideas with us regarding this project.

T.CHANDIRASEKAR(05C21)
N.G.KARTHIKEYAN(05C42)
S.KRISHNAPRIYA(05C47)

ABSTRACT

Semantic Solution Tracking System

eShelf – A Implementation of Semantic Search Engine

When you send a request to our smart search engine, it does more than just a keyword based lookup and return. Language processing helps the engine to uncover what you really meant to find.

The amount of computerized data continues to grow exponentially. There are huge collections of data available in the Internet. As more and more data on more and more topics is accumulated into the repositories, how to retrieve all the information on a topic readily, efficiently, and economically is our main concept. Adding intelligence that will assist in the search will be of enormous use. Semantic search uses semantics or the science of meaning in language to produce highly relevant search results. Our goal is to deliver the information queried by a user rather than have a user sort through a list of loosely related keyword results. In this project we have implemented a semantic search engine with the help of Natural Language Processing (NLP) and Information Retrieval (IR) technology [1] which uses Web Ontology Language, Dictionary hookup and Ranking algorithm for search results that will eventually help the Web users to retrieve the relevant results they need.

Table of Contents

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF GRAPHS

LIST OF TABLES

INTRODUCTION

1. Introduction

As with the WWW, the growth of the Semantic Web will be driven by applications that use it. Semantic search is an application of the Semantic Web to search. Search is both one of the most popular applications on the Web and an application with significant room for improvement. We believe that the addition of explicit semantics can improve search. eShelf attempts to provide semantics to the search engine and improve traditional search results (based on Information Retrieval technology and Natural Language Processing) by using data from the Semantic Web.

Traditional Information Retrieval (IR) technology[1] is based almost purely on the occurrence of words in documents. Search engines like Google, augment this in the context of the Web with information about the hyperlink structure of the Web. The availability of large amounts of structured, machine understandable information about a wide range of objects on the Semantic Web offers some opportunities for improving on traditional search.

Before getting into the details of how the Semantic Web[3] can contribute to search, we need to distinguish between two very different kinds of searches.

- **Navigational Searches:**

  In this class of searches, the user provides the search engine a phrase or combination of words which s/he expects to find in the documents. There is no straightforward, reasonable interpretation of these words as denoting a concept. In such cases, the user is using the search engine as a navigation tool to navigate to a particular intended document. We are not interested in this class of searches.

- **Research Searches:**

  In many other cases, the user provides the search engine with a phrase which is intended to denote an object about which the user is trying to gather/research information. There is no particular document which the user knows about that s/he is trying to get to. Rather, the user is trying to locate a number of documents which together will give him/her the information s/he is trying to find. This is the class of searches we are interested in.

Example:

A search query like ``W3C track 2pm Panel'' does not denote any concept. The user is likely just trying to find the page containing all these words. On the other hand, search queries like ``karthi'' or ``jenny'', denote a person or a place. The user is likely doing a research search on the person or place denoted by the query.

As it is specified in our base paper [1], we have defined rules in the OWL(Web Ontology Language)[2] which provides the semantics for our application to do research search.

Semantic search attempts to improve the results of research searches in 2 ways.

- Traditional search results[6] take the form of a list of documents/Web pages. We augment this list of documents with relevant data pulled out using our engine from the repository. For example a search for “class not found exception in java” might get augmented with “how to set the class path”, “types of run time exceptions”.
- The search phrase in Research Searches typically denotes one (or occasionally two) real-world concepts. We believe that it might be useful for the text retrieval part of the search engine to have an understanding of these concepts denoted by the search phrase. Understanding the denotation can

help understand the context of the search, the activity the user is

trying to perform, drive expectations on the categories of documents (pertaining to the object) likely to exist, etc.

We have implemented a semantic search engine[5] with the help of Natural Language Processing (NLP) and Information Retrieval (IR) technology which uses Web Ontology Language, Dictionary hookup and Ranking algorithm for search results that will eventually help the Web users to retrieve the relevant results they need.

eShelf – AN OVERVIEW

2. eShelf – An overview

eShelf is completely implemented on Visualstudio2008 with Dot net framework 3.5, MS SQL server 2005. Our app has two parts. First one is C sharp class library (eShelf.Support) which works as the heart of our application. Second one is asp.net pages (eShelf.Web) for UI based results and membership and authentication maintenance.

2.1. eShelf.Web

It contains the Web UI part of our application. It maintains ACL (Access Control List) and Searching controls. For maintaining ACL we have three type of users in our application. Admin user has full control over the app. He can view all the registered users with some statistics. He can be able to delete any unwanted user. Uploader user has access for only some specific files associated with him. He can be able to upload the files into the repository. While uploading the files, he must enter some description related to the file, which is stored in the database as the metadata. Regular user doesn't have any control over the application. He can just use the engine for searching. All type of user has access to it. It doesn't need any authentication. Then for the search part there is a crawler which initiates the

catalog builder in the beginning. All the aspx pages are hooked up

with the master page to enforce some security.

2.2. eShelf.Support

It acts as the heart of our application. When a user enters a search keyword, the crawler first checks whether this keyword is present in the catalog file. If catalog file is not there in the specified path, then it will start building the catalog. The catalog builder goes to the repository and parses through every file regardless of file type. It gets all the objects out of the files after doing some language processing works like stemming, stop words, go words and stores the state of those objects in a storage medium. We use .dat file extension for this. In medium trust we can store it in xml file too. Inside the serialized file we have specified the word and the file id (in which file the word is) and the position of the word in the file (It is used in displaying the results). Using a persistence class, the objects in the serialized medium made persist. After building the catalog, when the user enters the search keyword the spider goes through the OWL (Web Ontology Language) parser first. In our application, the owl file and associated parser is fully customized for placement related keywords. When it identifies a similar word in the owl file, it goes through its subclasses and gets those elements. If the given

word is available in the subclass itself then it gets its parent

element and also its siblings. And all those results put up in a list. After that the spider goes through the wordnet dictionary and gets the synonyms, meaning and the type of the word from the dictionary. It pushes all those synonyms inside the list. After that it goes through the catalog and finds where the word is. After doing these things, our ranking algorithm works. The first priority given to the objects from the OWL file and then same priority to the keyword based search and also the wordnet dictionary. Based on that it will display the search results. Using this, definitely a user can get what he really wants. It is proven with the statistics.

Figure1. eShelf – Home Page

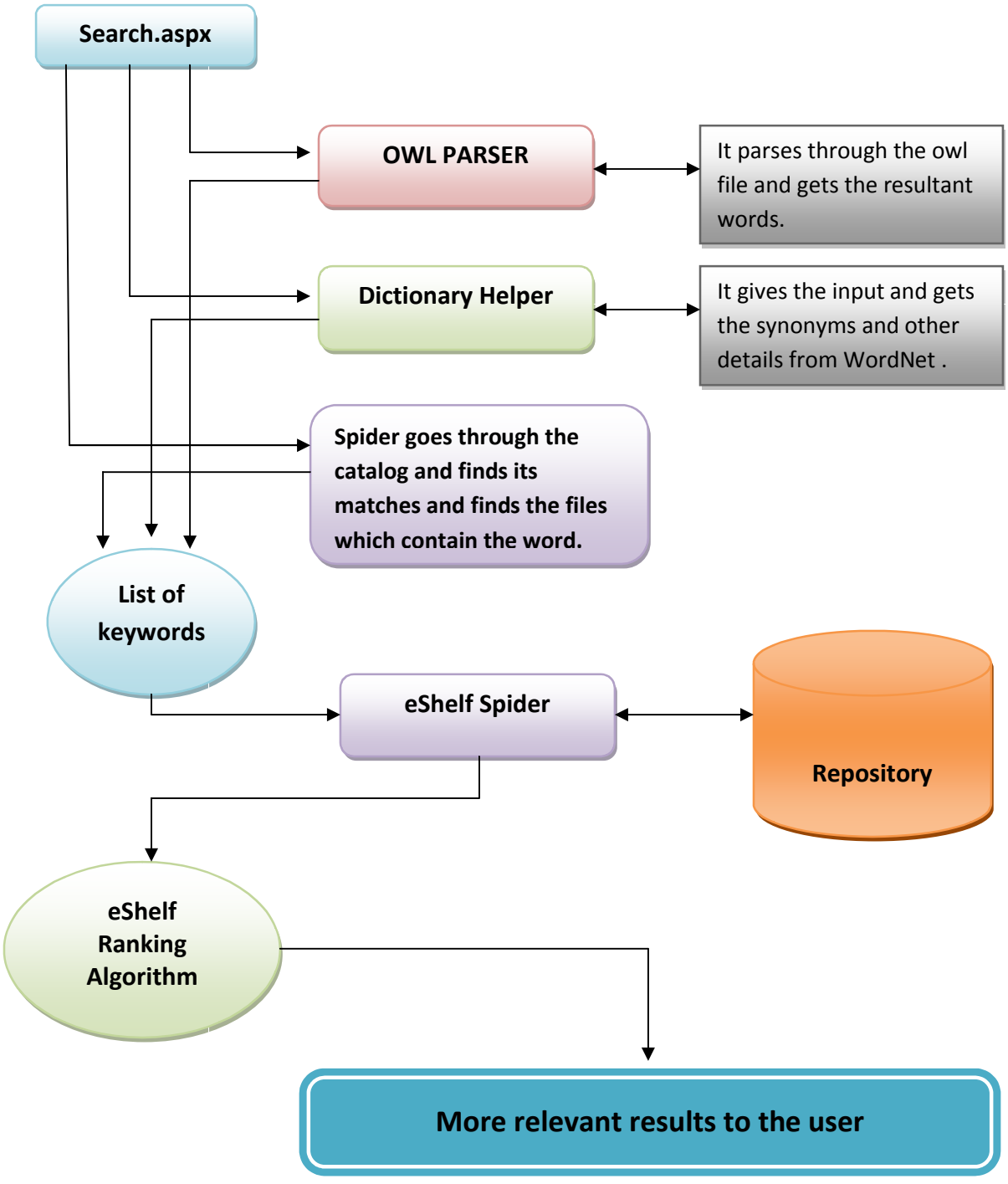


Figure2. eShelf – Architecture

2.3. eShelf – Architecture

The above figure resembles the web architecture of eShelf and the flow of its execution. Apart from ordinary search engine here, three more things involved. First one is, OWL Parser which uses Web Ontology Language for enabling semantics into our application. It has rules associated with every file. The second one is Integration of WordNet dictionary. If we give an input keyword to the dictionary, it gives the meaning, type of the word and its synonyms with examples. This is included into our project to extend the searching functionality. Next we have a specialized ranking algorithm, which has a great part in determining the order of results. The first priority is given to the OWL rules. Then keyword based results and WordNet results have the same priority. Spider crawls through out all the files to obtain the results. Apart from these things, we have membership and authentication controls. To help in search keywords, we have Ajax auto suggest option. Users can also upload files into the Repository.

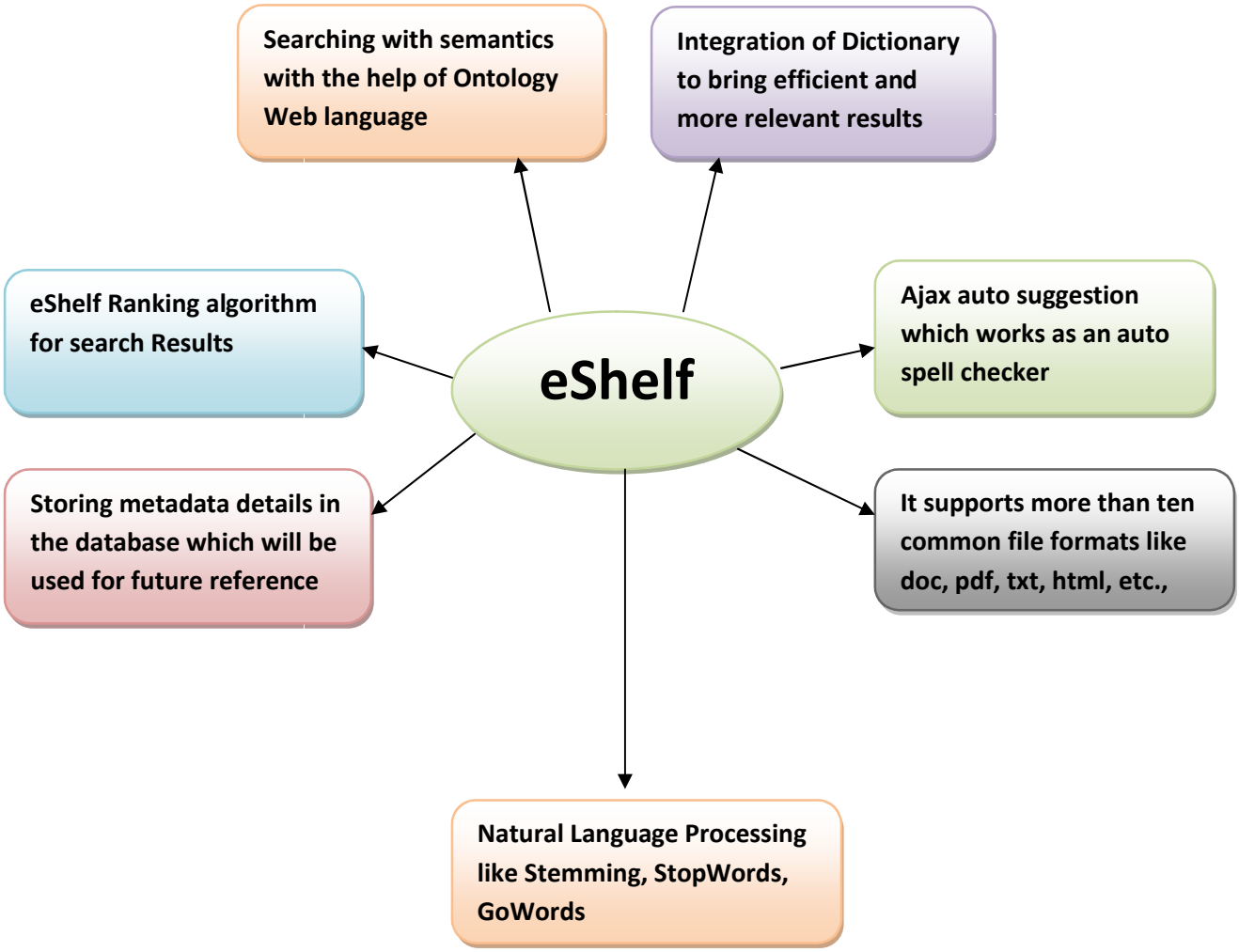


Figure3. Uniqueness of eShelf

HOW eShelf WORKS

3. How eShelf works

3.1. UI part of eShelf

3.1.1. User registration

If a user wants to upload files into the repository, he must register first. By clicking the uploader link, one can find the registration page. By entering all those details he can become a user of eShelf. Password must contain at least one alpha numeric character. He can also have privilege to change the password. Password recovery also has been done. He becomes a valid user.

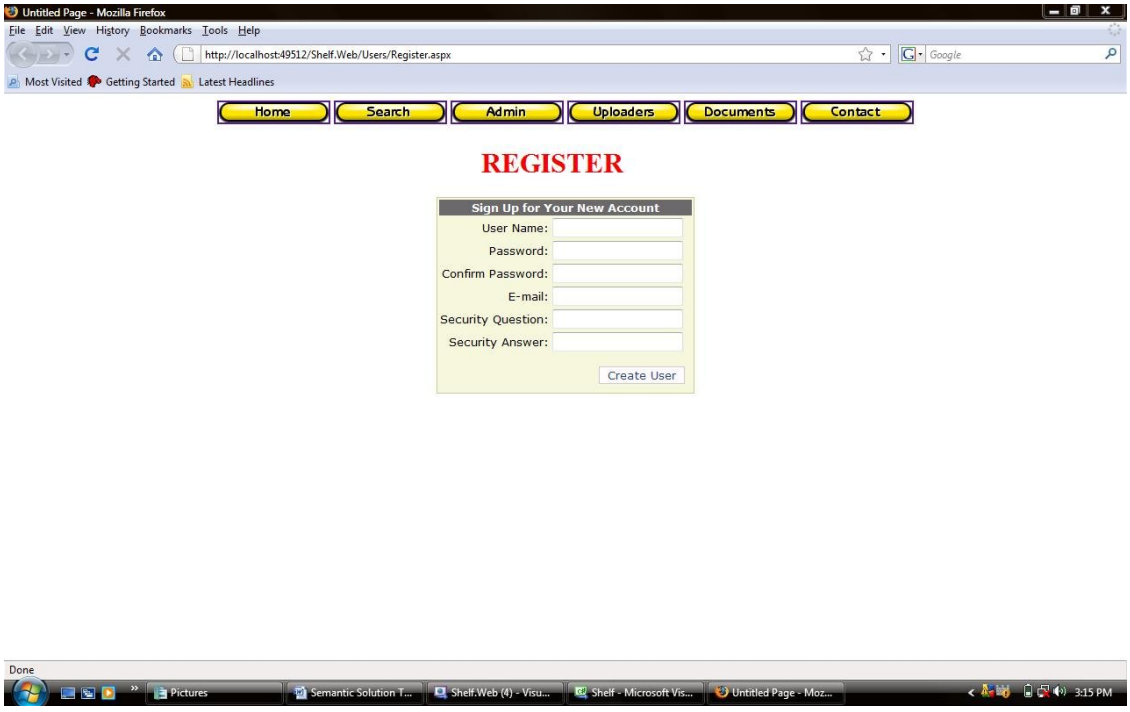


Figure4. New User Registration Page

3.1.2. Uploading files to the repository

Our repository path is specified in the web.config file. After a successful login of a valid uploader, he can upload the files he wants. Inside that he needs to specify a valid file name, category, file description and the file that he is going to upload. After validating the form, the user will be allowed to upload. All those details that the user enters will be stored inside the database. Those details will be used as metadata. It is stored for future references. It can be considered as manual rules associated with every file.

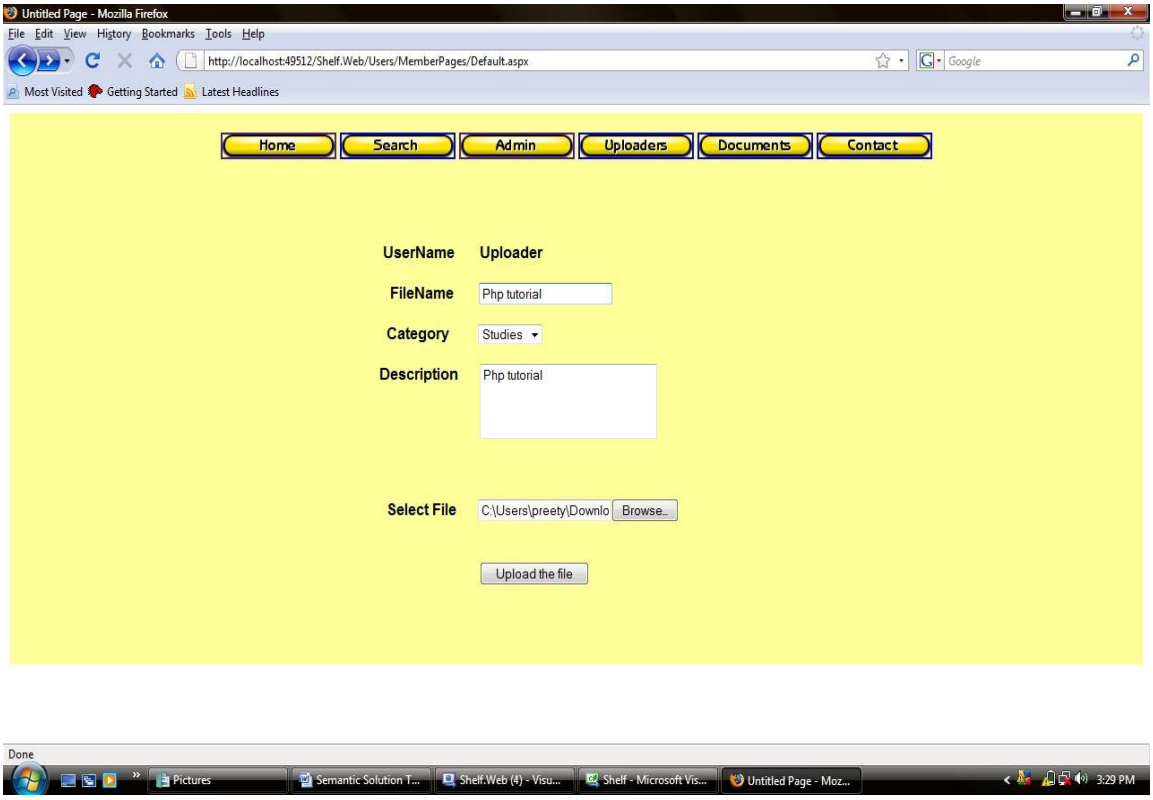


Figure5. File Uploading to the Repository

3.1.3. Maintaining ACL

ACL refers to access control list. Each type of user will be having different type of permissions and accessibility. Three types of roles are enabled for them. Three types of user are here in eShelf

- Admin - Have full control over the app
- Uploader - Can upload files to repository
- Regular user - Can only do searching

Admin role has access to all the files and controls in our app. Uploader role can do uploading files and searching. Regular user role can do searching alone. He doesn't have access to all other files.

We have shown the list of tables for maintaining the membership and authentication. There are a lot of stored procedures also there to maintain this.

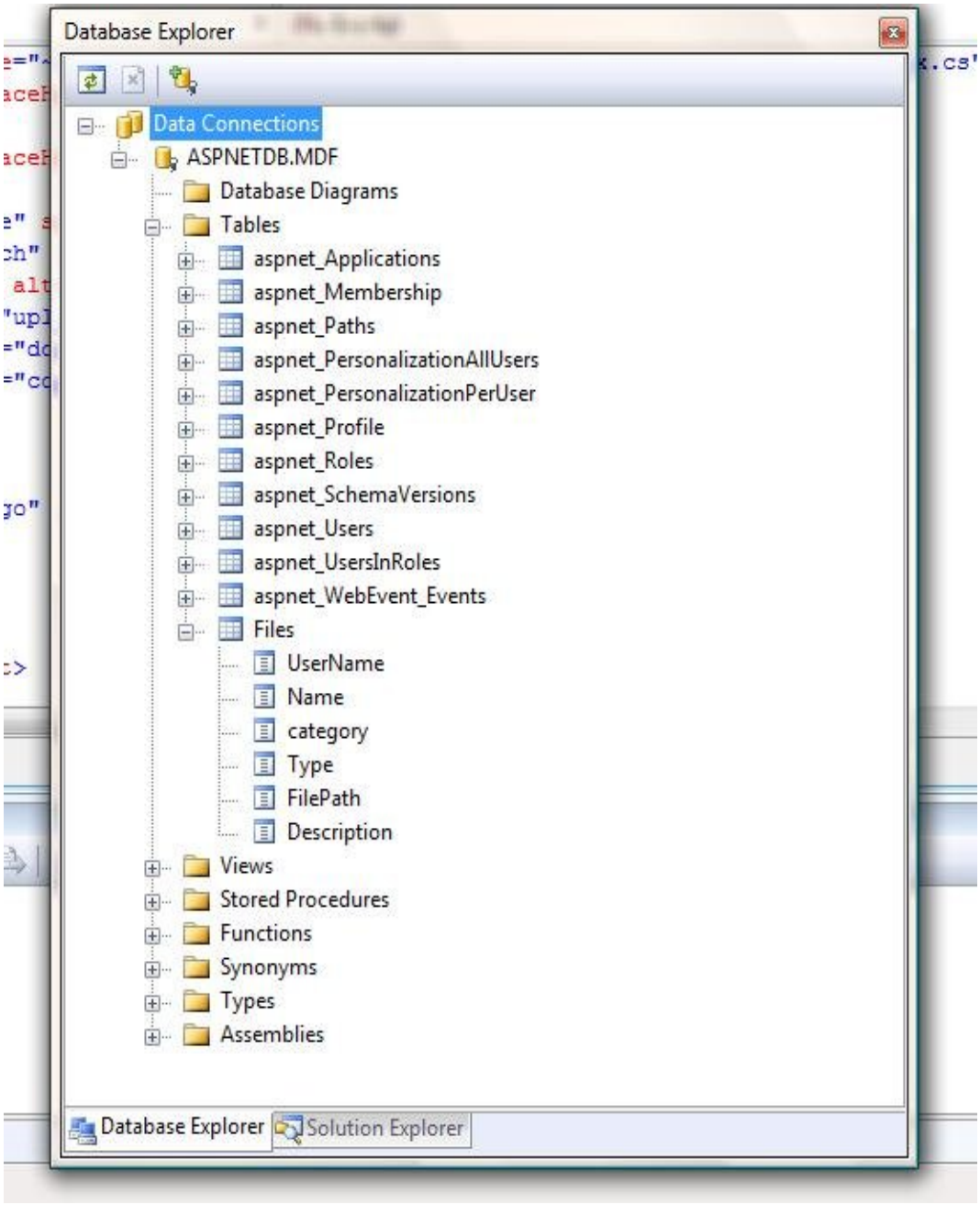


Figure6. eShelf – Database Schema

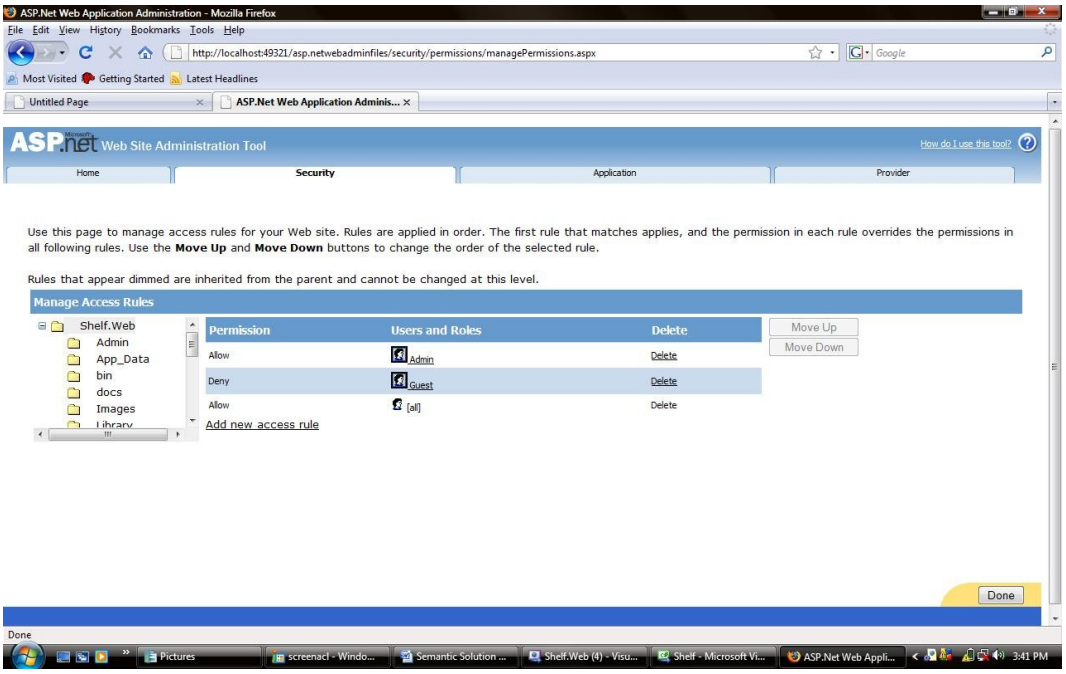


Figure7. Access rules and Roles maintenance

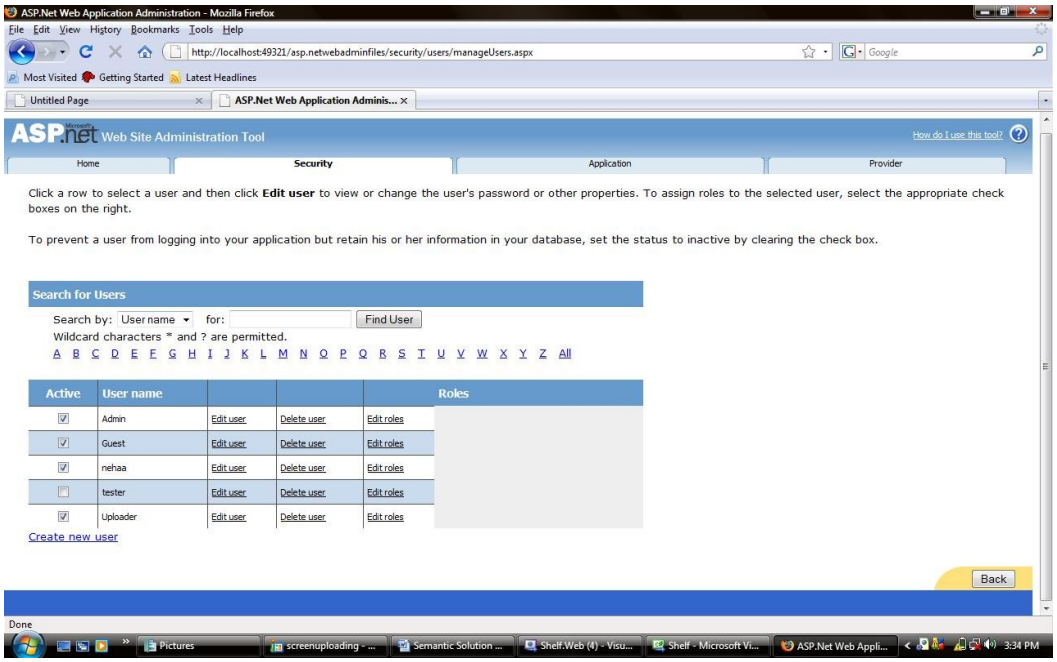


Figure8. User and their Role maintenance

3.1.4. Admin services

Admin role has full control over the application. Admin can view all the registered users and what are all the files each user has uploaded with some statistics. He can also delete the unwanted users. Admin also can be able to upload files. He can be able to modify the OWL file. He can add some more rules to it.

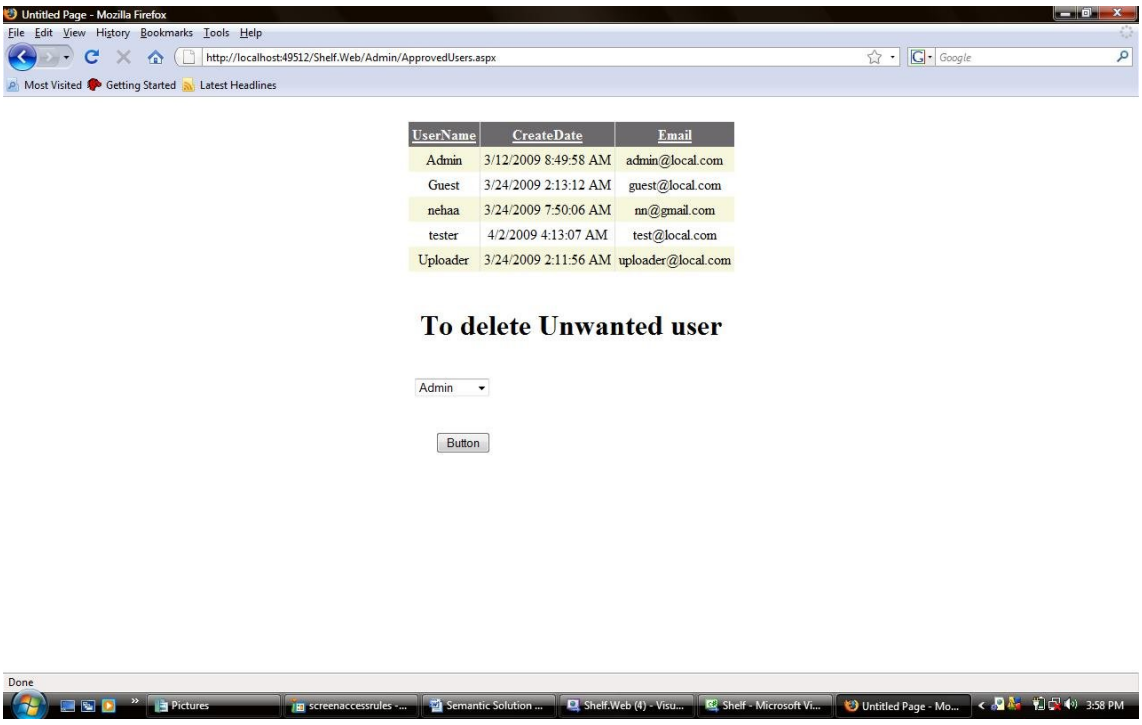


Figure9. Admin user view for control over the App

3.2. Catalog building

3.2.1. Supported file extensions

eShelf spider can index html, doc, docx, txt, rtf, ppt, pptx, xls, xlsx, xml, pdf and also jpeg images. Every file extention has different type of indexing mechanism.

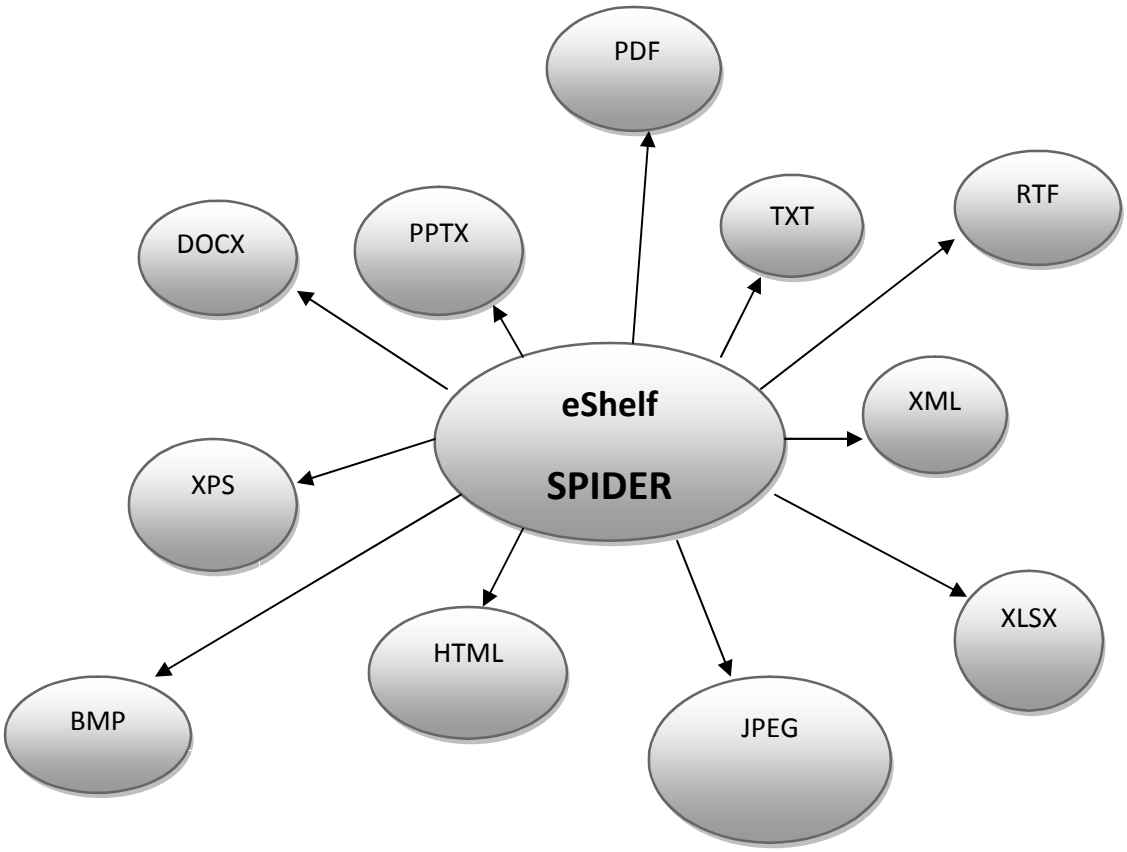


Figure10. File formats that eShelf can Index

3.2.2. IFilters – Indexing Service

IFilter refers to Index Filtering. The IFilter interface scans documents for text and properties (also called attributes). It extracts chunks of text from these documents, filtering out embedded formatting and retaining information about the position of the text. It also extracts chunks of values, which are properties of an entire document or of well-defined parts of a document. IFilter provides the foundation for building higher-level applications such as document indexers and application-independent viewers. Implement this interface if you are providing a filter to extract information from a proprietary file format so that the text and properties can be included in the index. Full-text search engines like Indexing Service call the methods of this interface to extract text and property information for creating an index.

These IFilters are mainly used in Building the catalog. When the catalog builder starts running this IFilter goes through every document that is compatible with eShelf and indexes words out of it before doing the natural language processing.

3.3. Natural Language Processing

3.3.1. Stemming – Porter stemmer algorithm

The Stemmer class transforms a word into its root form[7]. The input word can be provided a character at time. We have implemented the six steps of his algorithm.It deals with six steps of processing.

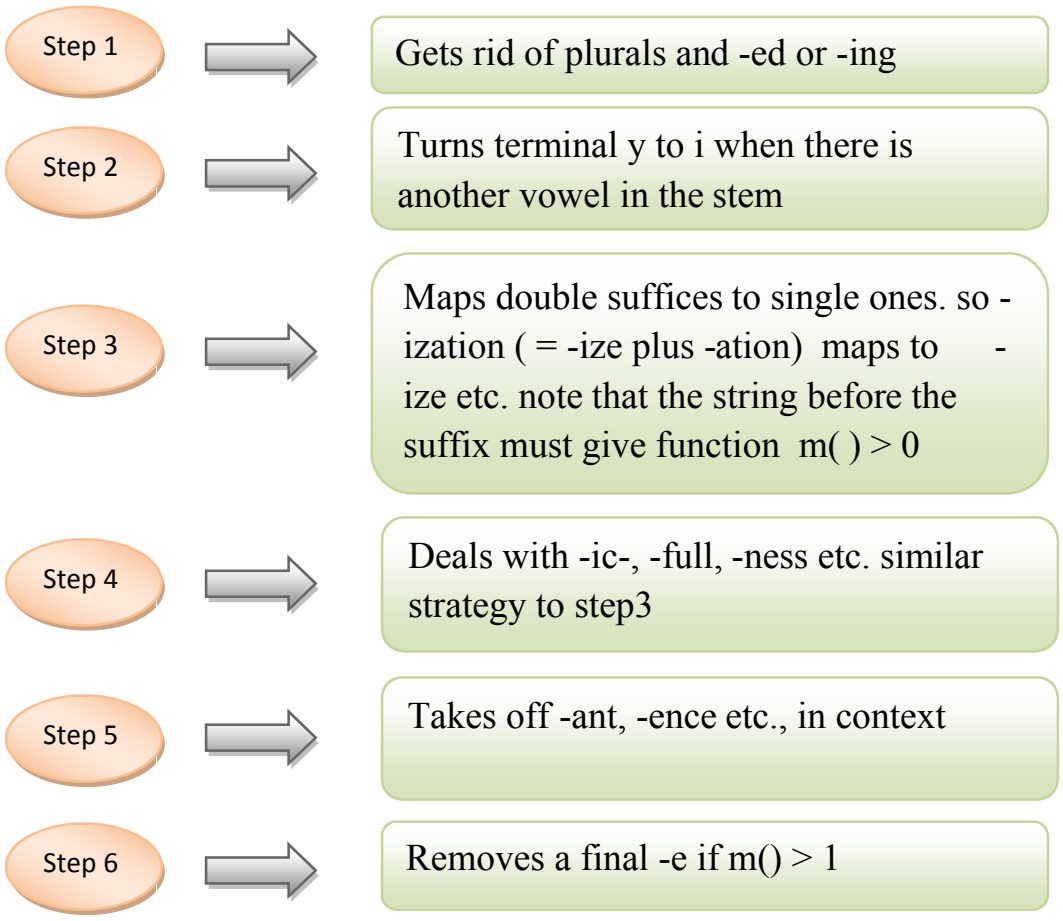


Figure11. Six steps of Porter Stemming Algorithm

Examples for Stemming process:

Step	Pattern	Example
1.	sses → ss ss → ss (m>0) eed → ee (*v*) ed → (*v*) ing → (*v*) Y → I	Caresses → caress Caress → caress Agreed → agree Plastered → plaster Motoring → motor Happy →happi
2.	(m>0) ational → ate (m>0) tional → tion (m>0) enci → ence (m>0) izer → ize (m>0) abli → able (m>0) alli → al (m>0) ator → ate (m>0) alism → al	Relational → relate Conditional → condition Valenci → valence Digitizer → digitize Conformabli →conformable Radicalli → radical Operator → operate Fuedalism →feudal
3.	(m>0) icate →ic (m>0) ative → (m>0) alize → al (m>0) iciti → ic (m>0) ical → ic (m>0) ful → (m>0) ness →	Triplicate → triplic Formative → form Formalize → formal Electricity → electric Electrical → electric Hopeful → hope Good ness → good
4.	(m>1) ic → (m>1) ful → (m>1) ness →	Gyroscopic → gyroscop Hopeful → hope Goodness → good
5.	(m>1) ant → (m>1) ence →	Irritant → irrit Inference → infer
6.	(m>1) e → (m=1 and not *o) e →	probate → probat rate → rate cease → ceas

Table1. Examples for stemming process

3.3.2. Stopwords

Stop words sometimes known as stopwords or noise words, is the name given to words which are filtered out prior to, or after, processing of natural language data. We have listed a set of words as stop words in StopWords.cs. These words will not be indexed in the catalog. If the spider finds a word which is listed in the stop words it will just skip the word. Basically the stop words will be of 3 or 4 characters. Some of the stop words we have listed: the, and, that, you, this, for, but, with, are, have, was, out, not.

3.3.3. Gowords

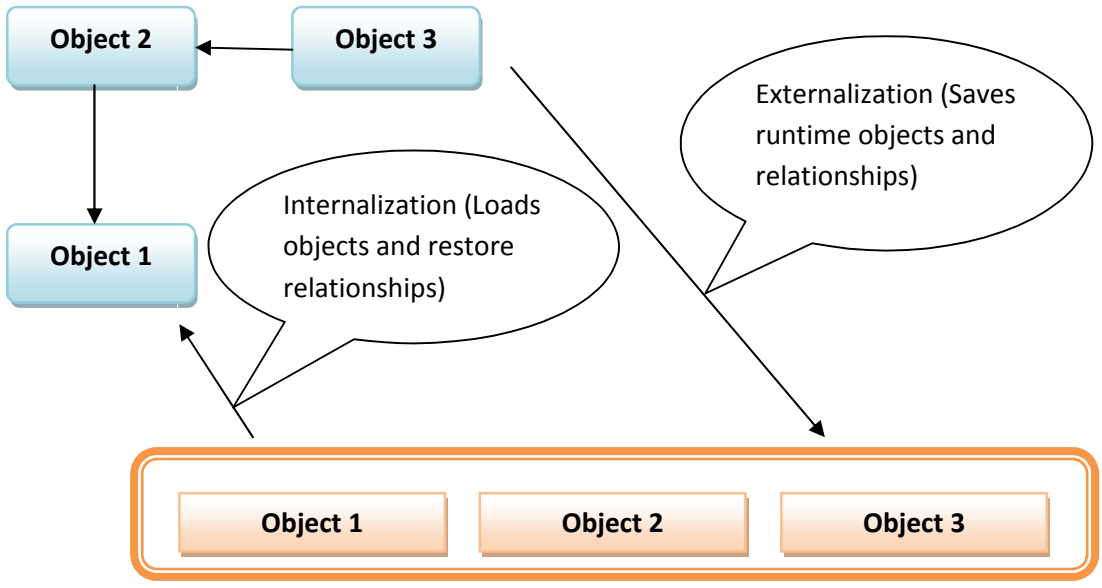
Go words, is the name given to words which are skipped out prior to, or after, processing of natural language data. If a word is listed in the go words list, it will not go through any trimming process even though the word has any special characters or alpha numeric characters. It will be indexed as it is. Some of the listed go words: c#, vb.net, asp.net, c++.

3.4. Binary serialization

Serialization can be defined as the process of storing the state of an object to a storage medium. During this process, the public and private fields of the

object and the name of the class, including the assembly containing the class, are converted to a stream of bytes, which is then written to a data stream. When the object is subsequently deserialized, an exact clone of the original object is created.

eShelf has all indexed words as binary objects. These objects must be stored in a storage medium. In medium trust we can store it in xml file using xmlserializer class. While retrieving the information from that, all serialized objects will be deserialized to get the appropriate results.



(Objects residing in the memory)

(Objects saved in a file in XML or binary format)

Figure12. Binary Serialization

3.5. Catalog Objects

Index contains ‘link’ between each word and the URL of documents that contain it[8]. The number of times that word appears or where that words in appears is lost during the indexing process. Significantly of the index to store more data the structure is altered as, for each word-document pairing, we also store the positions of that word in the source document. For example: After parsing out punctuation and whitespace each word is assigned an index, with the first word given position zero and each subsequent word adding one. We also store the complete text of the document and can therefore extract any given part of the text.

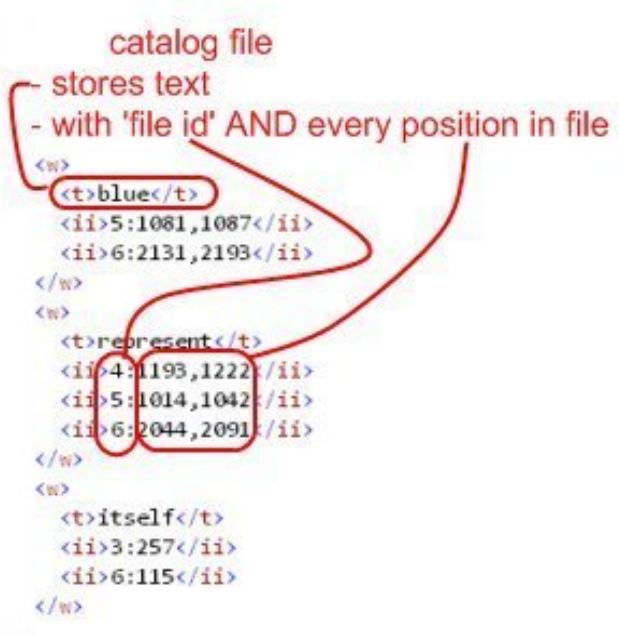


Figure13. Objects in an XML catalog file

The figure shows how the catalog is built.

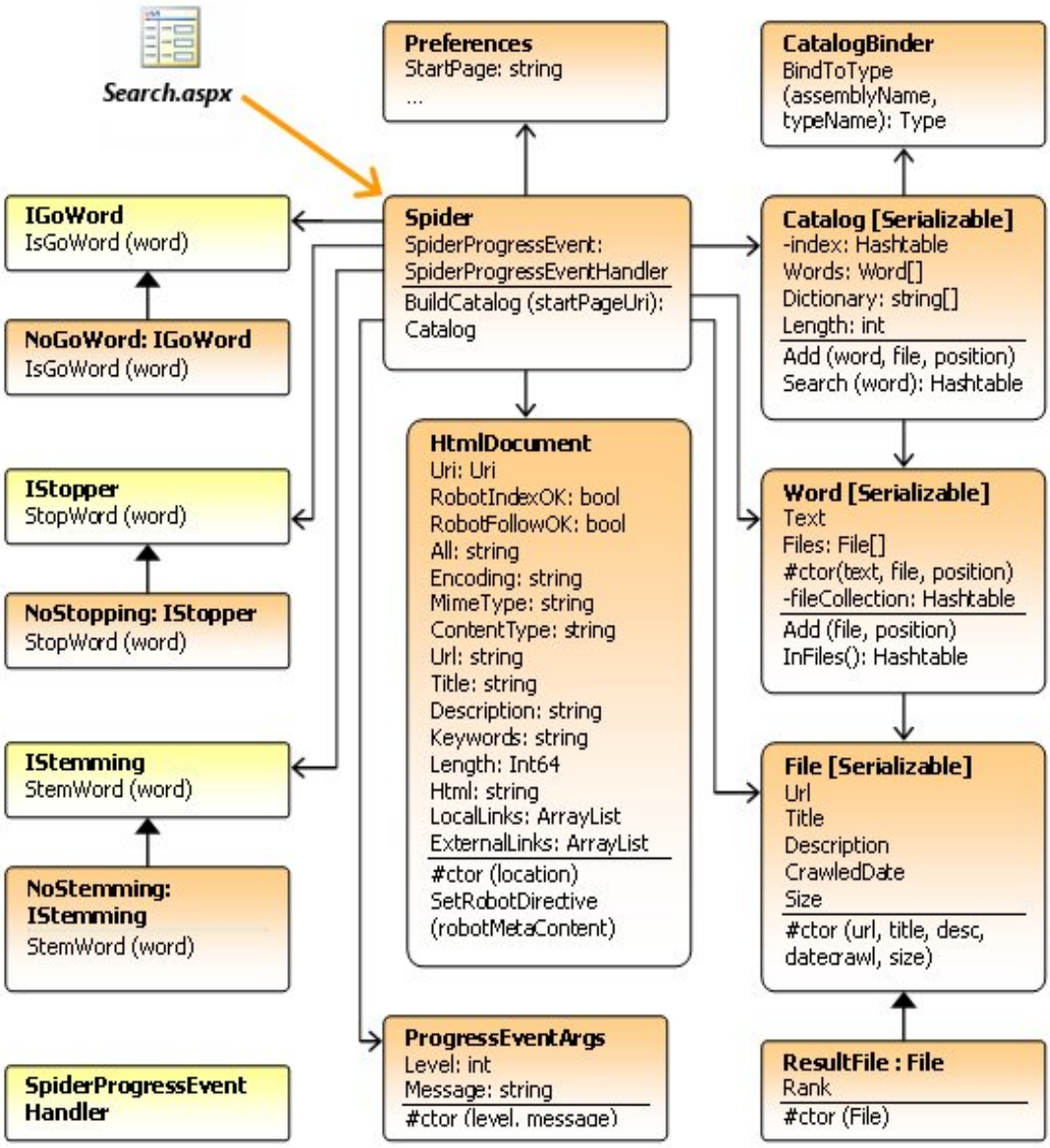


Figure14. Catalog building process

3.6. Object persistence

After serializing all the objects it will be stored in a medium as a binary file. Those objects must be made persist. It should remain in the same state. Using this persistence class all the serialized objects is stored in a .dat file. In medium trust we can store it in an xml file. Here is the figure shows our binary catalog file. Since it is in binary format it will not be understandable. We have highlighted some of the indexed words after doing natural language processing.

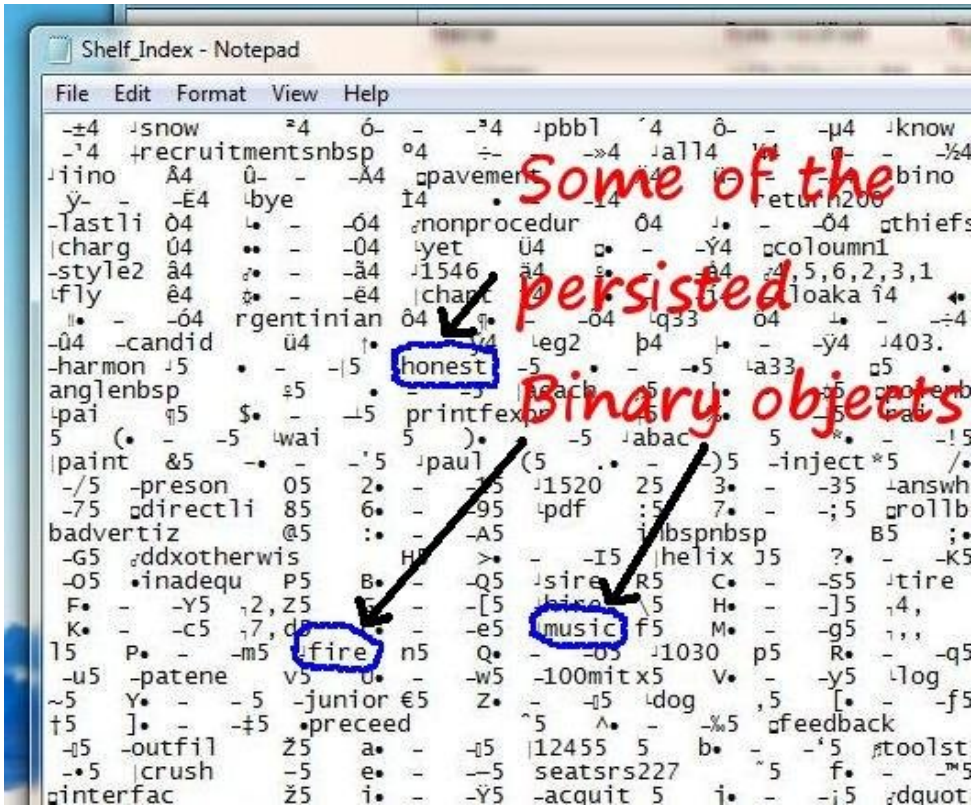


Figure15. Objects in a Binary file

eShelf SEARCHING

4. eShelf Searching

4.1. OWL Parser

Considering semantic search, Web Ontology language provides semantics. Web Ontology Language is a special type of file formatting for RDF documents. In eShelf we have customized our searching for the placement papers domain. Our owl file contains search terms related to placement domain and every search term will be considered as parent classes. When user searches with a keyword, the owl parser get it as input and search for the occurrence. If the key word is found as a parent node, then it will return its children. If it is identified as a child node then it will return its parent node along with its siblings. The results will be put up in a sorted list.

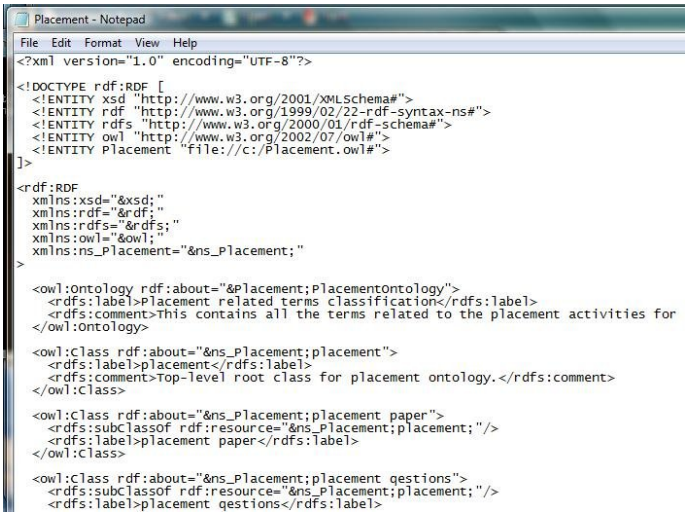


Figure16. View of the OWL file

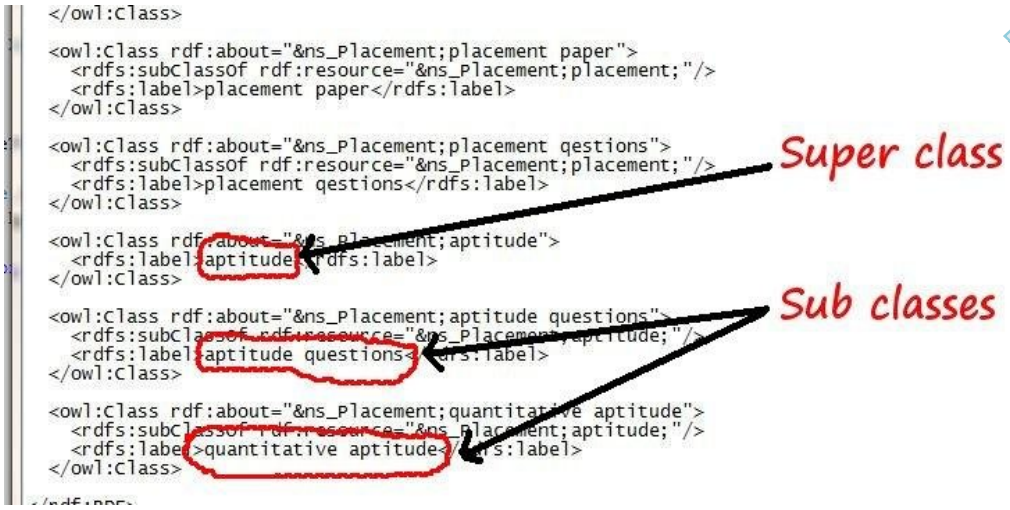


Figure17. OWL file – Denoting classes

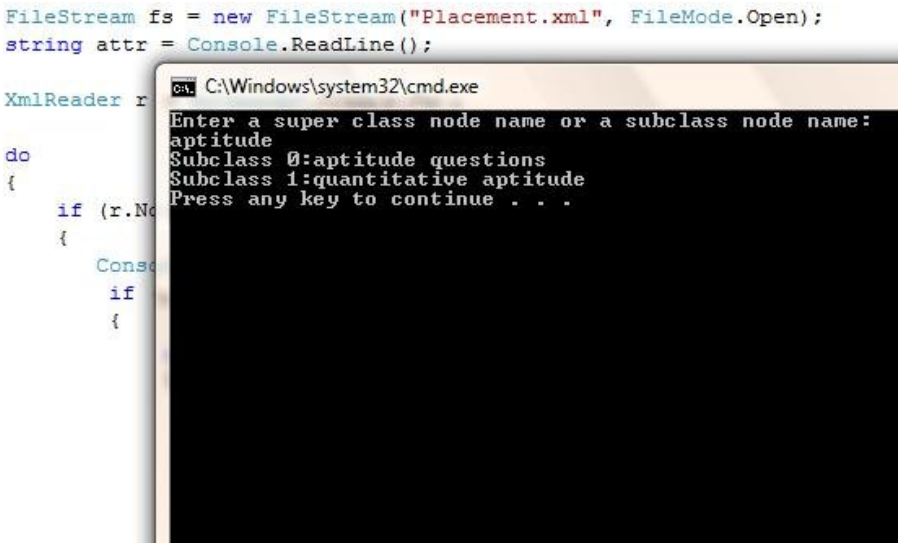


Figure18. XML Parser – Console application

4.2. Dictionary hookup

To make our search more efficient, we have integrated WordNet dictionary with eShelf. WordNet dictionary[4] is a popular online dictionary developed by Princeton University. It contains almost all the words in English in binary format. When you give an input word to the dictionary, it will return its meaning, type of the word (whether it is a noun, verb, adverb and adjective) and synonyms of that word. These results also added with the sorted list.

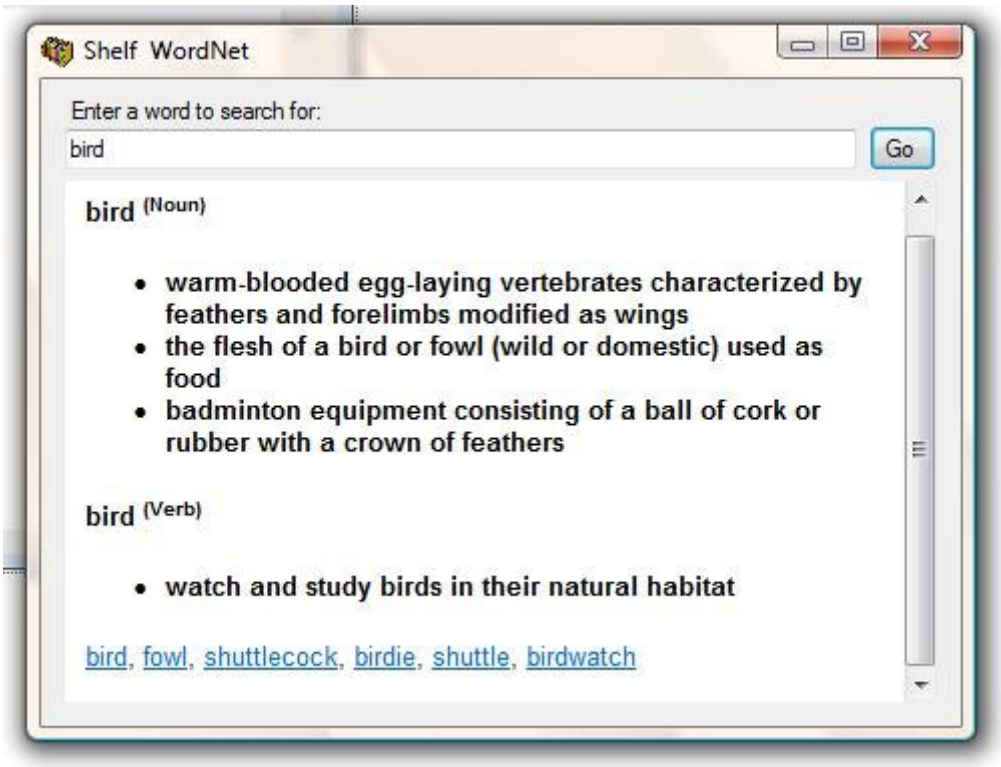


Figure19. Standalone WordNet Dictionary App

4.3. Ajax auto suggestion

eShelf is completely customized for a single domain, for Placement training related searching. Instead of adding a customized spell checker, we have integrated Ajax auto suggestions option. When a user started typing the first character of the searching term, it will display all the possible combination of the words starting with the same character.

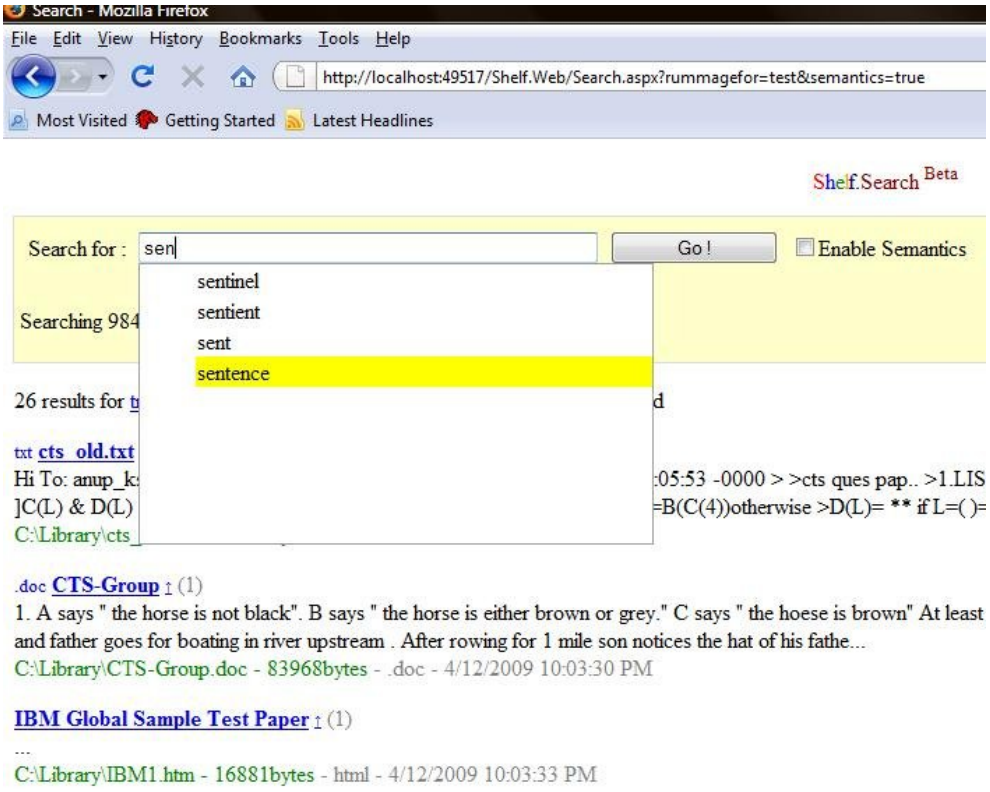


Figure20. Ajax Auto Suggestion

4.4. Keyword based Searching

This is similar to traditional search engine. It searches for the input query string in all the documents present in our repository using the catalog and returns the results. But implementing a keyword based search is also not an easy task. There are a lot of constraints for that. First we must consider whether we are going to have our own Repository or we are going to search in the Internet.

If it is our own Repository means then probably we may know the size of it. The indexing or cataloging mechanism we are going to handle will bean important part of it and file formats it is going to support. And what are all the natural language processing mechanism we are going to handle to index those documents.

Suppose we are going to perform the search in the Internet then we have to consider all those things specified above. We need an efficient crawler which parses through every page. Here we don't know the size of the data. Probably it may be very huge. After crawling through the pages we must cache all those data and perform some operation over that data.

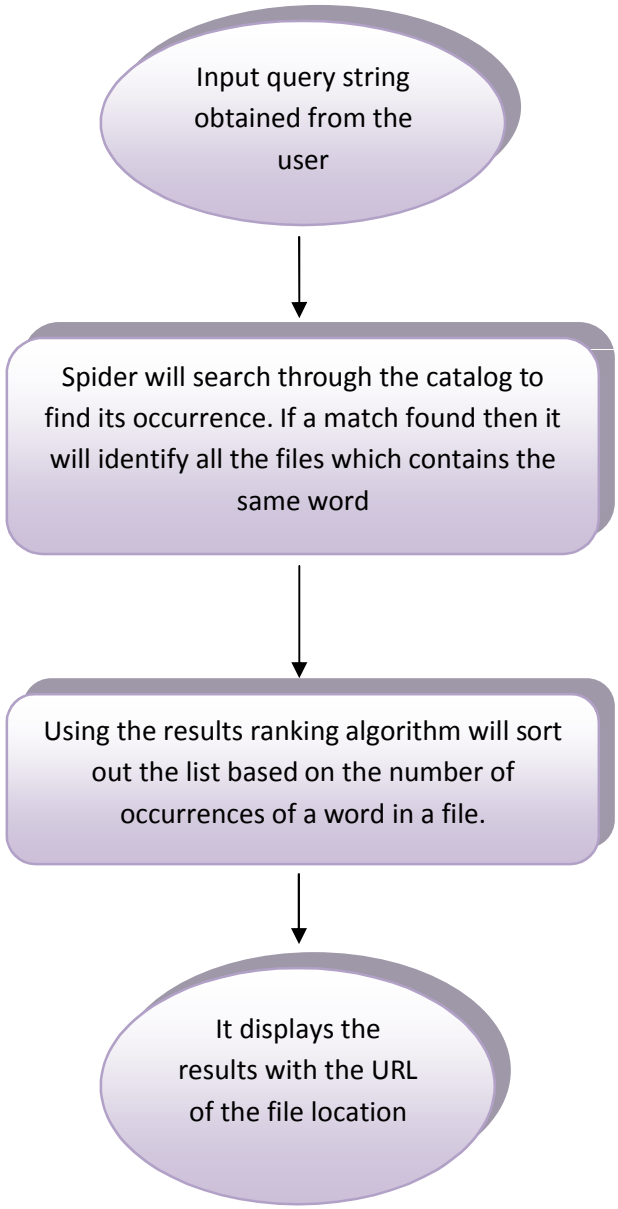


Figure21. Keyword based searching

The above figure specifies the spider searches from our Repository of files and not from the Internet. It is customized for eShelf.

4.5. Ranking algorithm

We have implemented our own ranking algorithm for eShelf. In our algorithm, the first priority given to the semantics and rules from the OWL file and then same priority to the keyword based search and also the WordNet dictionary. Based on that search results will be displayed. Using this, definitely a user can get what he really wants with extremely high probability. It is proven with the statistics collected personally from student associates.

Search for anything using your favorite crawler-based search engine. Nearly instantly, the search engine will sort through the millions of pages it knows about and present you with ones that match your topic. The matches will even be ranked, so that the most relevant ones come first.

Of course, the search engines don't always get it right. Non-relevant pages make it through, and sometimes it may take a little more digging to find what you are looking for. But, by and large, search engines do an amazing job.

Unfortunately, search engines don't have the ability to ask a few questions to focus your search, as a librarian can. They also can't rely on judgment and past experience to rank web pages, in the way humans can.

So, how do crawler-based search engines go about determining relevancy, when confronted with hundreds of millions of web pages to sort through? They follow a set of rules, known as an algorithm. So, in eShelf we follow our own set of algorithm.

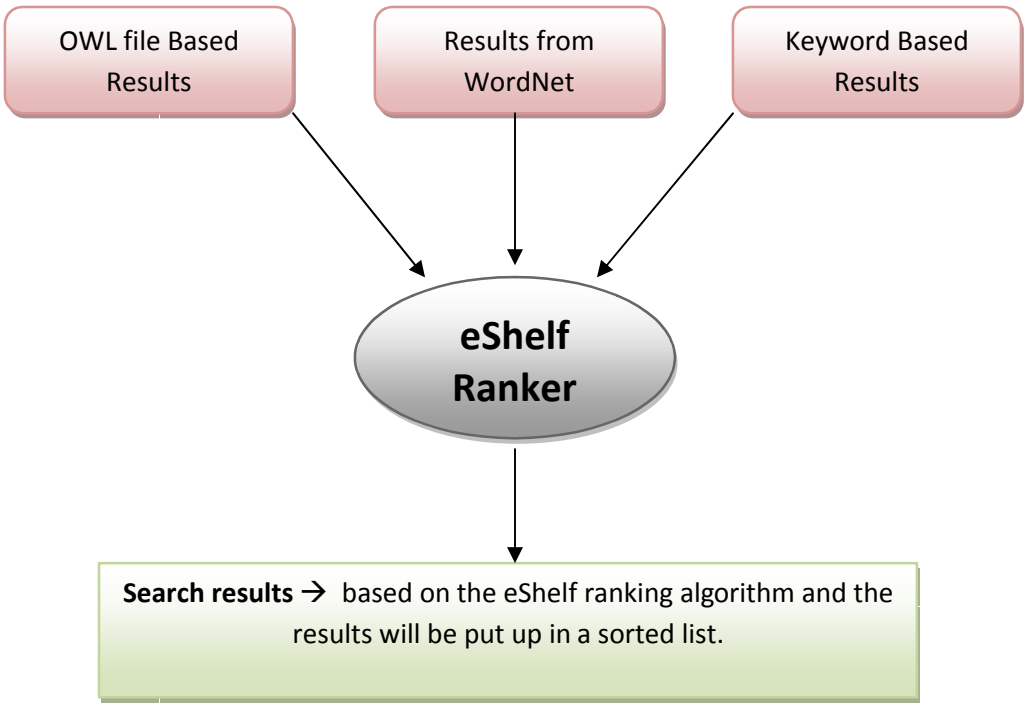


Figure22. eShelf Ranking Algorithm

4.6. Search results

When the engine gets an input query, it invokes the owl parser and dictionary finder to get the results and also invokes the spider to search inside the catalog. After getting all the results they will be sorted according to the ranking algorithm and then the results will be displayed to the user with pagination.

For example, let us take the word “pattern” as our search term.

The following figure shows how ordinary keyword based search engine works. The spider shows number of words it has indexed. It is shown below the search text box. Our spider has indexed 9849 words from the repository. When a used search for a word “pattern” it goes through the catalog and finds out the files containing the word and the URL of those files. Those things will be put up in a list. After that all those pages will be sorted based on the frequency of the given word or the number of occurrence of the word in the file. And the results will be sent to the user. The type of the word also displayed along with every result.

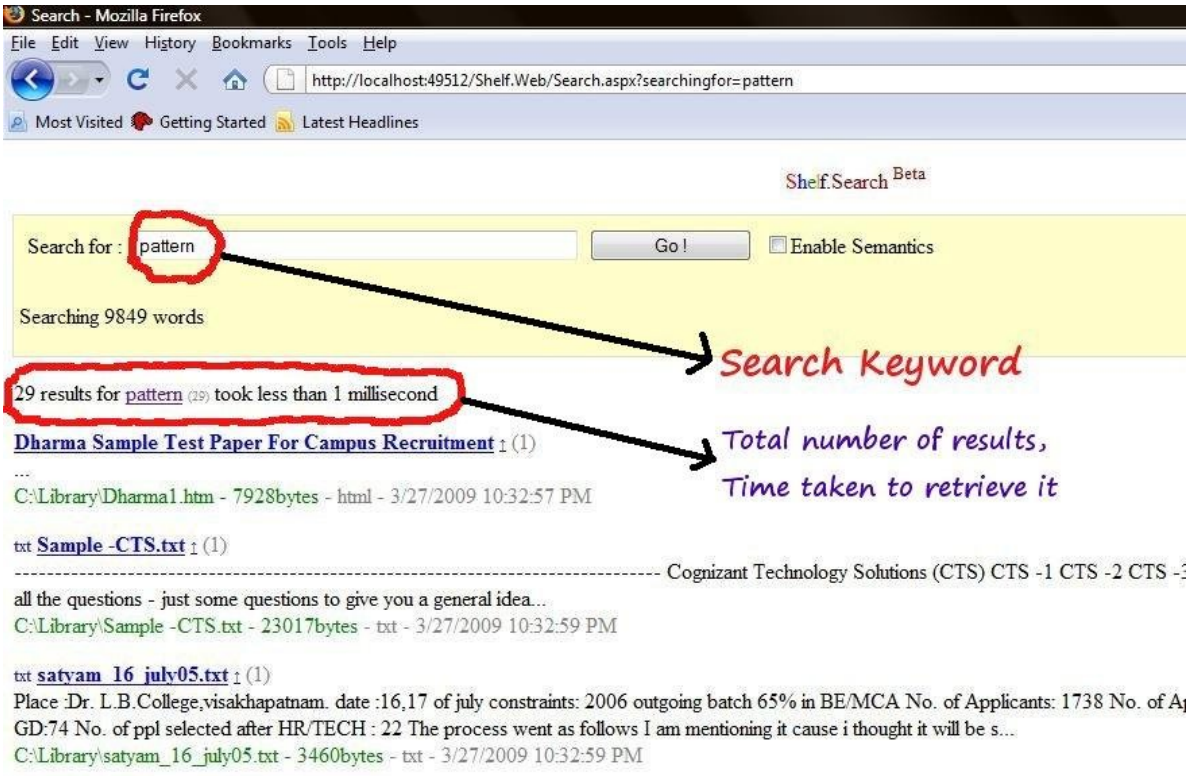


Figure23. Ordinary keyword based search

The result page also contains the time taken to retrieve the result. Mostly it won't cross a millisecond since it searches from our own Repository. When we consider the Internet it may take more time to search. We can achieve the same speed there too if we have a sophisticated caching mechanism.

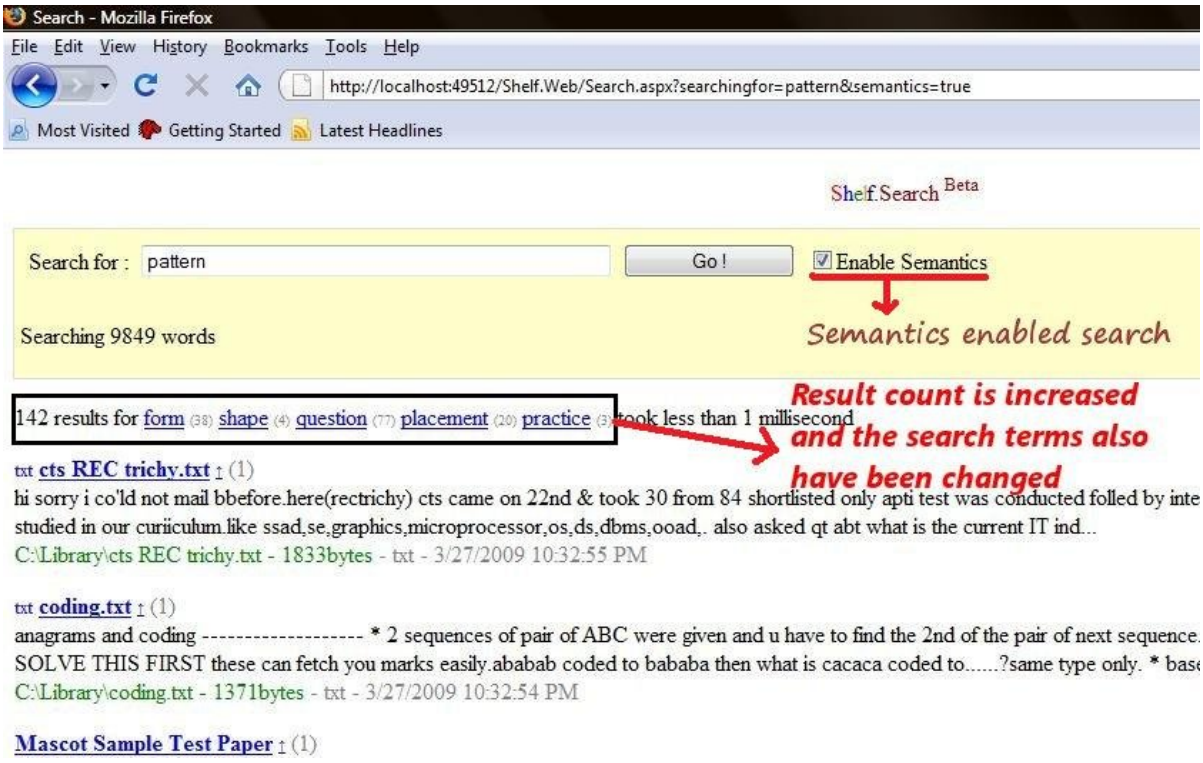


Figure24. Semantics enabled search

The figure above shows the results from semantics enabled search. There is a drastic change between the ordinary search and semantics enabled search. Even you can see the changes in the results too. Ranking algorithm makes it happen to bring out more relevant result. Search terms also changed due to OWL based search and dictionary hook up.

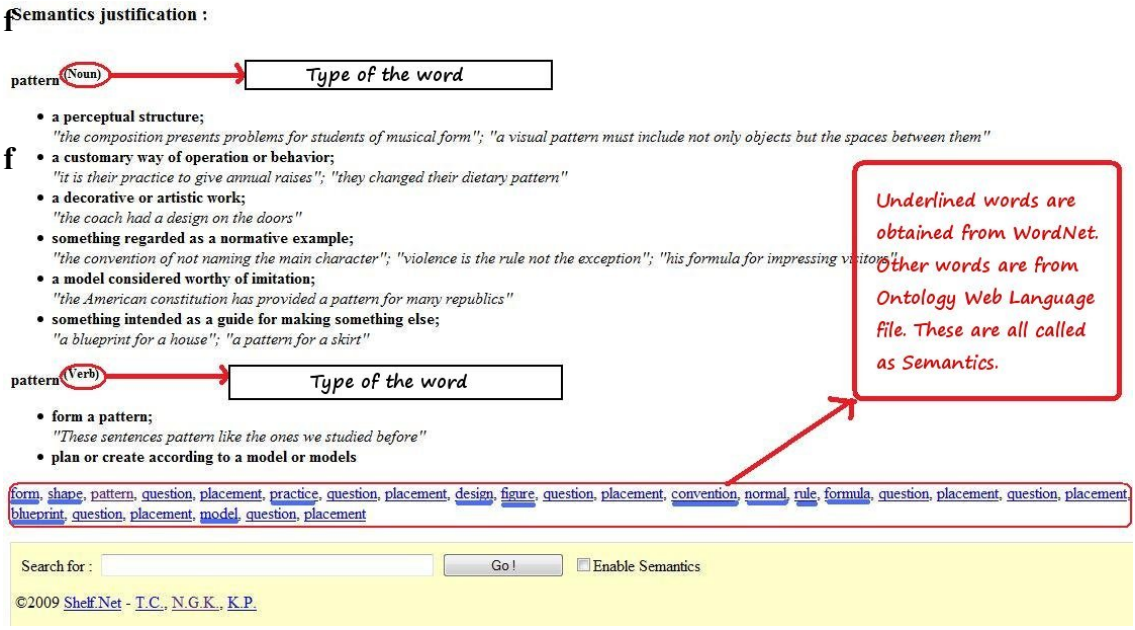


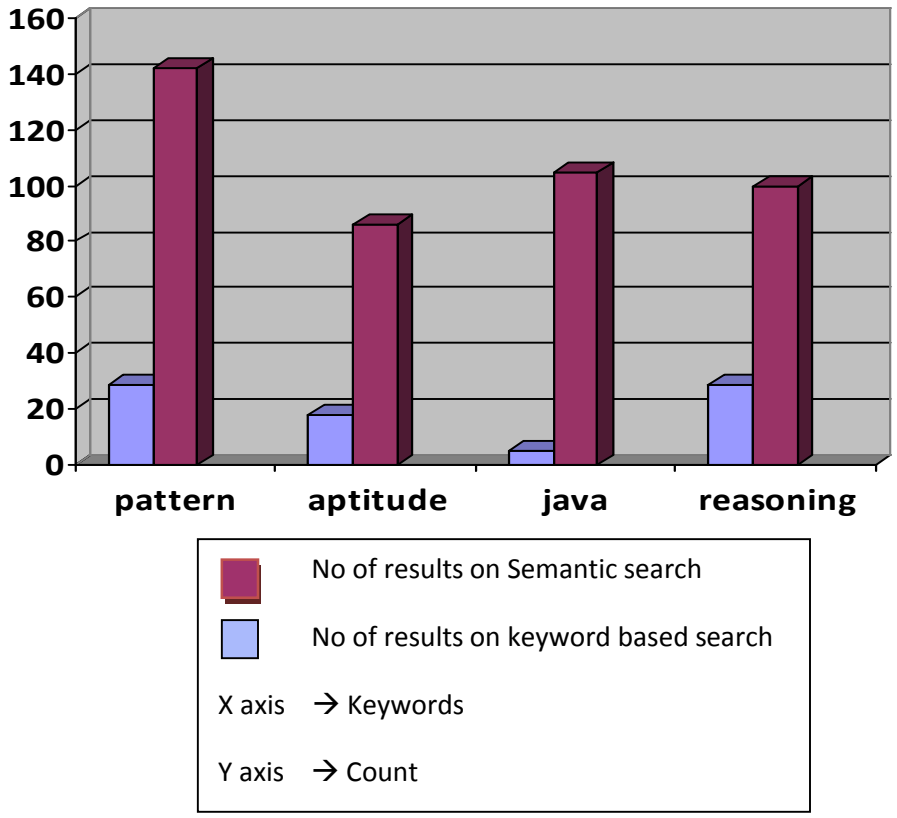
Figure25. Obtained Semantic Results

These are the semantic justifications for the given keyword. The dictionary gives the meaning and also some examples using that word. The underlined words are obtained from the Dictionary and other are from the OWL results.

PERFORMANCE ANALYSIS

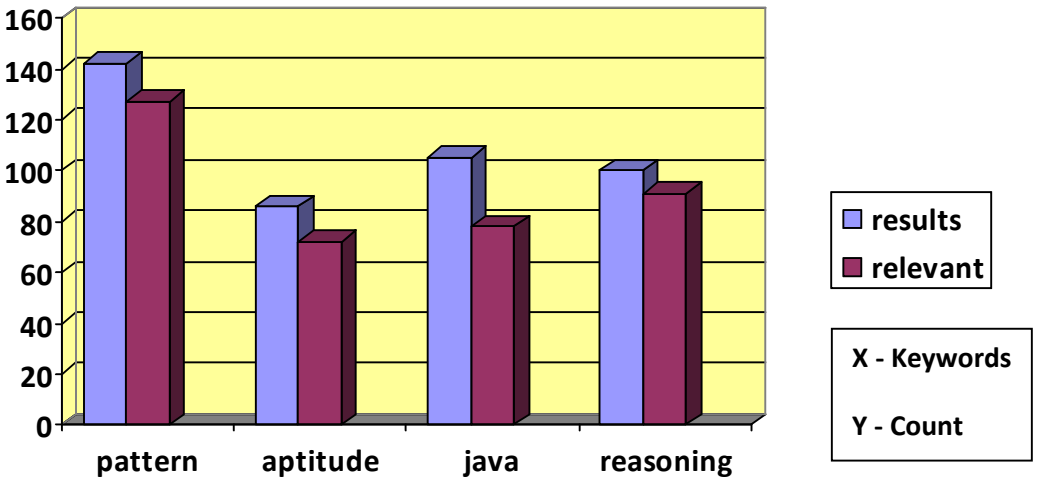
5. Performance Analysis

To differentiate eShelf from existing search engines, we need to have performance based analysis. So at the first, we have represented the difference between ordinary keyword based searching and semantic searching. For that we have taken 4 key words: pattern, aptitude, java, and reasoning. The results are shown in the graph. Obviously semantic search produces more results. But it doesn't mean, we have got the relevant results.



Graph1. Keyword Based Search Vs Semantic Search

To analyze whether we have obtained the correct result, we have performed Relevancy Analysis. We have taken the same 4 words as example. We have analyzed all the results of it. At the end of the result, we have identified that eShelf obtains 92% percent of more relevant results than ordinary keyword based search.



Graph2. Obtained result Vs Relevant result

It doesn't mean that the remaining 8% results are useless. They will be useful if we come out of the particular domain based searching. Because in eShelf we are mainly focusing on a single domain: student placement. Otherwise those 8% results also will be useful.

CONCLUSION AND FUTURE WORK

6. Conclusion and Future Work

In this project, we have tried to enable a search engine to give more relevant result. Every search engine follows different algorithm. But the efficiency matters. We have tried our own algorithm. We have analyzed it with other search engines too. eShelf is somewhat similar to a digital library because it searches from its own Repository. It is customized for a single domain. Apart from an ordinary keyword based search we have implemented owl file based semantic search, dictionary hookup, ranking algorithm, storing Meta data of each file in to our database, Ajax auto suggestion, Natural Language processing like Stemming, StopWords and GoWords. It also supports for many type of documents.

For further development on eShelf, we can do a complete search instead of doing a domain based search. We can run crawler to index documents which will run through all the specified paths[5]. We can also connect it to external Repositories and we can store the Meta data in the database. While doing that we must also look at the crawler speed. It needs more efficient algorithm to parse through the files. We can also generate rules dynamically while crawling. It needs more sophisticated algorithm to do that.

REFERENCES

7. References

[1] Changping Hu, Yang Zhao, " **An Ontology based for service in digital library"** Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on Volume , Issue , 21-25 Sept. 2007

[2] Deborah L. McGuinness, Richard Fikes, James Hendler, and Lynn Andrea Stein. `` DAML+OIL: **An Ontology Language for the Semantic Web** ". In IEEE Intelligent Systems, Vol. 17, No. 5, pages 72-80, September/October 2002.

[3] Andreas B. Ebarhart **Building a domain aware eSupport system** Research associate at International University, Germany

[4] **WordNet – An Electronical Lexical DataBase** implemented by Princeton University, Princeton, NJ Ref: http://wordnet.princeton.edu

[5] **Semantic mash up search engine** Author: Haytham El-Fadeel Researcher in Computer Sciences hfadeel@acm.org

[6] **How Search Engines work** January 1997 issue of Microsoft Internet Developer Author: TizianaPerinotti Copyright © TGP Consulting 1995-2002. Ref: http://www.tgpconsulting.com/articles/mind.html

[7] **The Porter Stemming Algorithm**
Ref:http:// tartarus.org/~martin/PorterStemmer

[8] **Catalog building process** Reference:
http:// pluralsight.com/blogs/craig/archive/2004/07/08/1580.aspx

APPENDIX

eShelf Review

Here we have listed some of the important terms we have used in our project with the references from the Internet.

Binary Serialization

Serialization can be defined as the process of storing the state of an object to a storage medium. During this process, the public and private fields of the object and the name of the class, including the assembly containing the class, are converted to a stream of bytes, which is then written to a data stream. When the object is subsequently deserialized, an exact clone of the original object is created.

When implementing a serialization mechanism in an object-oriented environment, you have to make a number of tradeoffs between ease of use and flexibility. The process can be automated to a large extent, provided you are given sufficient control over the process. For example, situations may arise where simple binary serialization is not sufficient, or there might be a specific reason to decide which fields in a class need to be serialized. It is shown in the diagram clearly [Figure12. PageNo: 25].

Web Crawler or Spider

A Web crawler is a computer program that browses the World Wide Web in a methodical, automated manner. Other terms for Web crawlers are *ants,* automatic indexers, bots, and worms or Web spider, Web robot.
A Web crawler is one type of bot, or software agent. In general, it starts with a list of URLs to visit, called the seeds. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to

visit, called the crawl frontier. URLs from the frontier are recursively visited according to a set of policies.

OWL

The **Web Ontology Language** (**OWL**) is a family of knowledge representation languages for authoring ontologies, and is endorsed by the World Wide Web Consortium. This family of languages is based on two (largely, but not entirely, compatible) semantics: OWL DL and OWL Lite semantics are based on Description Logics, which have attractive and well-understood computational properties, while OWL Full uses a novel semantic model intended to provide compatibility with RDF Schema. OWL ontologies are most commonly serialized using RDF/XML syntax. OWL is considered one of the fundamental technologies underpinning the Semantic Web, and has attracted both academic and commercial interest.

In eShelf we have tried our own methods to implement a semantic search engine with the help of some references. It can be extended in future as a complete search engine for the Internet. But there is a lot of work to do better work. We will improve it further with a lot of features to make it a complete search engine.