SEMANTIC BASED SEARCH ENGINE

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

YAMUNA SAI MUKKAMALA

IIT Roorkee - CDAC, NOIDA,
c-56/1, "Anusandhan Bhawan"
Sector 62, Noida-201 307

JUNE, 2004

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this dissertation titled "**SEMANTIC BASED SEARCH ENGINE**", in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Information Technology**, submitted in **IIT, Roorkee-CDAC campus, Noida,** is an authentic record of my own work carried out during the period from June 2003 to June 2004 under the guidance of **Dr. Poonam Rani Gupta**, Associate Professor, CDAC, Noida.

I have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

Date: 28·06·2004
Place: Noida

[illegible]
(YAMUNA SAI MUKKAMALA)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

[illegible]

Date: 28·06·2004
Place: Noida

(Dr. POONAM RANI GUPTA)
Associate Professor,
CDAC, Noida.

ACKNOWLEDGEMENTS

I hereby take the privilege to express my deep sense of gratitude to **Prof. PREM VRAT**, Director, Indian Institute of Technology, Roorkee, and **Mr. R.K.VERMA,** Executive Director, CDAC, Noida for providing me with this valuable opportunity to carry out this work. I am very grateful to **Prof. A.K.AWASTI**, Programme Director, **Prof. R.P. AGARWAL**, course coordinator, M.Tech (IT), IIT, Roorkee and **Mr. V.N.SHUKLA**, course coordinator, M.Tech (IT), CDAC, NOIDA for providing the best of the facilities for the completion of this work and constant encouragement towards the goal.

I express my sincere thanks and gratitude to my Guide **Dr. POONAM RANI GUPTA,** Associate Professor, CDAC, Noida, for her inspiring guidance and sustained interest throughout the progress of this dissertation.

I am thankful to **Mr. R.K.SINGH,** senior project engineer, CDAC, Noida and **Mr. MUNISH KUMAR**, project engineer, CDAC, Noida for providing necessary infrastructure to complete the dissertation in time.

I owe special thanks to my friends, all of my classmates and other friends who have helped me formulate my ideas and have been a constant support. And I also thank my family members for their moral support.

[illegible]

(YAMUNA SAI MUKKAMALA)

Enroll. No. 029030

TABLE OF CONTENTS

ABSTRACT

Internet search engines are special sites on the web that are designed to help people find information stored on other sites. There are differences in the ways various search engines work, but they all perform these basic tasks- Searching the Web based on important words to build their databases; keeping an index of the words they find, and where they find them; allowing users to look for words or combinations of words found in that index. Most of the today's search engines perform key word search causing search irrelevance and incompleteness of the search. To over come these problems semantic search is used. To perform semantic search two approaches may be considered:

- Search based on Semantic Web
- Search based on Generic Relevance

Semantic Web Search: The Semantic Web comprises recent work to improve the relevance of search results by integrating semantics into Web pages, which focuses primarily on defining standards based on an ontology or a metadata model and syntax like the World Wide Web Consortium's resource description framework. Semantic frameworks are arguably an interesting approach for a single organization or a limited topic, especially where particular functionality exists to restrict the ontology and thus allow semiautomatic support for query processing. The approach has also proven useful for navigating subject-specific sites.

Generic Relevance Search: Relevance is typically person dependent, so personalization becomes important feature of this search. User constructs a profile based on his search intention. Using his profile the search is expanded or narrowed down to a particular topic. This approach is very useful for some restricted domains. To find the user relevant meaning of the given query terms WordNet is used.

Chapter 1

INTRODUCTION

An effective navigation through Internet is difficult to imagine without search engines. The World Wide Web environment can be compared to a large encyclopedia of more than a hundred million pages without a table of contents. Now over four million pages respond even to the word combination "search engine" in Internet, which indicates an information explosion on the information search problem. No single search engine was indexing more than about 16% of Web sites, and all major search engines combined were covering only about 42% of the Web.

It is also found that Web is growing by about 7.3 million Web pages a day. About 84% of all pages are based in the USA. It is estimated that about 30% Web pages are copied or mirrored, or are very similar Analysis of 200 million Web pages has shown that these pages use 1.5 billion hyperlinks and that the Web is made up of four components. A central core (about 56 million pages) is well connected, an 'in' cluster (44 million pages) contain pages that point to the central core but cannot be reached from it, an 'out' cluster (44 million pages) contains pages that can be reached from the core but do not link to it, 'tubes' (44 million pages) connect to either 'in' or 'out' clusters but not to the core, while some 17 million pages are completely unconnected. Web graph structure is interesting in itself, but it can also be used for designing crawl strategies on the Web

Web sites have extremely variable information value since they are written by individuals with any conceivable background, education, culture, interest, and motivation, and in most cases these materials are neither edited nor reviewed. A Web page size may range from a dozen words to hundreds screens, and often contains grammatical mistakes and typos. Presented material can be obsolete, false or unreliable. Web content itself is unstructured, and multiple formats, different languages, dialects and alphabets are used.

Despite all its weakness, including a lot of poor, obsolete and unreliable material, Web includes a vast quantity of excellent material of all sorts, from practical information to professional and scientific material that cannot be found without appropriate tools.

Web search is also an exciting activity in itself, since search is one of the fundamental human activities. The more ordered and structured material is, it is less exciting. - The Web, with its heterogeneous and unstructured material, is therefore the perfect search area. Without software tools for searching and cataloguing of the huge and heterogeneous Web information space it would be absolutely impossible to find relevant information residing on the Web. The need for retrieval of information on the Web is so large that the major search engines receive about 15-20 thousand queries per minute.

What users need are powerful, easy to use search tools capable to provide a moderate number of appropriately ranked relevant answers In the last few years numerous advancements were made in various areas of the World Wide Web search. This section presents analysis of major aspects of recently developed Web search methods and tools, as well as of the most important trends in Web search methods.

1.1 Subject trees

Both subject trees (directories) and search engines cope with a serious problem of how to follow a tremendous growth of the Web. Since Web links are incorporated into subject trees by human indexing teams, the size of the team determines the coverage of Web contents by the subject tree. Since Web today contains roughly two billion Web pages, biggest subject trees cover only the order of magnitude of one promile of the Web. Although this is a very small coverage, the value of subject trees is primarily in the quality of Web links selected by its editors.

1.2 Search Engines

These are based on information that is collected, sorted and analyzed automatically A software program, (known as a "robot", "spider" or "crawler") reads web pages, follows links between pages and sites, and collects information for later use. The information collected is analyzed into an "index" which is like a huge database of all the sites the crawler visited.

When you search using one of these search engines, you are really searching the index. The results of the search will depend on the contents of the index, which are themselves based on the contents of each web page, including the title of the page, text, etc.

Changing the information on your web pages will usually (eventually) cause the entries in the index for your site to change, although this will often happen once the search engine's software has seen your new content. Modifying your web page can make your site relevant for more/less/different searches, and change how and where you appear in the search results listing.

1.3 Database search

Growing number of databases from various areas are available by means of the Web. However, search engines often cannot distinguish between the simple Web page and the entrance to one or more huge databases. Information contained in databases are hidden and cannot be retrieved by search engines, hence the name "Invisible Web" for databases accessible from the Web. Databases typically include structured data about specific topics like companies, restaurants, or locations of ATM machines. Most database resources are free, while some are fee-based.

1.4 Scientific information retrieval

Web is increasingly becoming a distributed collection of scientific literature where scientific papers can be rather easily retrieved and quickly accessed. Numerous researchers make their publications available on their homepages, and many traditional journals offer access to the whole text of papers on the Web. Some publishers allow their papers to be placed on the author's Web site, while other permit prepublication of articles on the Web. The main problem for search engines is in location of articles prepared in Postscript or PDF format, predominantly used formats for scientific publications.

Important step forward in enabling efficient and effective service for scientific literature on the Web was done by the NEC Research Institute team in developing the *ResearchIndex*. ResearchIndex is a search engine specifically designed for scientists, based on the combination of digital library and citation index. Citation indices index the citations in a paper so that the paper is linked with the cited articles. They enable scientists to find papers that cite a given paper, and also facilitate evaluation of articles and authors. The main problem with traditional citation indices is their high price related with manual effort necessary for indexing.

In order to avoid this problem NEC Research Institute team developed a digital library of scientific publications that create a citation index autonomously using Autonomous Citation Indexing (ACI), a system that doesn't require any manual effort. ACI allows literature search using both citation links and the context of citations. It retrieves PDF and postscript files and uses simple rules based on formatting a document to extract the title, abstract, author and references of any scientific paper it finds. ACI enables fast feedback by indexing items such as conference proceedings or technical reports. New papers can be automatically located and indexed as soon as they are published on the Web or announced on mailing lists.

The same team built a prototype digital library called CiteSeer. CiteSeer downloads papers from the Web and converts them to text, parses the papers to extract the citations and the context in which the citation were made, and then stores these information in a database. CiteSeer includes full-text articles and citation indexing and allows location of papers by keyword search or citation links. NEC Research has made the CiteSeer software available at no cost for noncommercial use. A demonstration version of *CiteSeer* indexes over 200.000 computer scientific papers, more than largest online scientific archives.

Retrieval of scientific data is a particularly complex problem, since it has to deal with widely distributed, heterogeneous collections of data. Scientific data form huge and fast growing collections of data stored in specialized formats (e.g. astronomic or medical data), and their retrieval requires powerful search tools. For this purpose a consistent set of metadata semantics, as well as standard information retrieval protocol supporting the metadata semantic is required. One technology developed for scientific information retrieval is *Emerge* the system based on XML-based translation engine that can perform metadata mapping and query transformation.

1.5 Trends

Several new approaches to search engines mechanisms were developed around the idea of exploiting rich Web hyperlinking structure for clustering and ranking of Web documents. *Clever* project, run by researchers from IBM, Cornell University and the University of California at Berkeley, analysis Web hyperlinks and automatically locates two types of pages: “authorities” and “hubs”. “Authorities” are the best sources of information on a particular broad search topic, while “hubs” are collection of links to these authorities. A respective authority is a page that is referred to by many good hubs, while a useful hub is a page that points to many valuable authorities. For any query

Clever first performs an ordinary text-based search using an engine like AltaVista, and takes a list of 200 resulting Web pages. This set of links is then expanded with Web pages linked to and from those 200 pages – this step is repeated to obtain a collection of about 3,000 pages. Clever system then analysis the interconnections between these documents, giving higher authority scores to those pages that are frequently cited, and higher hub scores to pages that link to those authorities. This procedure is repeated several times with iterative adjusting of authorities and hub scores: authority that has many high-scoring hubs pointing to it earns a higher authority score, while a hub that points to many high-scored authorities gets a higher hub score. The same page can be both the authority and the hub. A side effect of this iterative processing is that the algorithm separates Web sites into clusters of similar sites. Clever system is also used for automatic compilation of lists of Web resources similar to subject trees – these lists appear to be competitive with handcrafted ones.

One extension of this research is the *Focused Crawling* system, software for topical resource discovery. This system learns the specialization by examples and then explores the Web, guided by a relevance and popularity rating mechanism. The system selects Web pages at the data-acquisition level, so it can crawl to a greater depth (dozens of links away from the starting set of Web pages) when on the interesting trace. Focused Crawling[3] will be used for automated building of high-quality collection of Web documents on specific topic.

An interesting approach to Web search, sometimes called 'surfing engine', is developed by *Alexa* service. When an Alexa user navigates to a Web page, Alexa service retrieves data about this page and presents them to the user. These data (or metadata) includes information on who is behind the site the user is navigating to, how often is this site updated, how fast this site responds to requests, what is its popularity ranking, etc. All these information helps user to decide whether the site is what he is looking for. Alexa service also retrieves information from its servers to suggest the user some other pages (Related Links) that might be of interest to him. To find Related Links Alexa service uses the paths of the collective Alexa community, information about clusters of similar Web sites (found by prior analysis), analysis of texts on individual Web pages, etc. To avoid the "Not Found" message Alexa service checks its archive to see whether it can find an archived version of the page.

Visual information retrieval becomes increasingly important as the amount of image and video information in different computing environments continues to grow at an extremely fast rate. This kind of retrieval is very specific and complex, and even formulating the query is not simple since text queries are unable to express adequately the query requirement, making the query processing inefficient. Readers interested in this field are recommended to read two introductory papers.

Traditional search in the batch mode leads to the situations where users may miss recently added Web pages because of the slow update of search engines data bases (crawlers often revisit the same Web site after one or two months). This led IBM to develop the *Fetuccino* software, an interesting combination of traditional search in batch mode and dynamic search in real time. In the first stage some traditional batch search engine is exploited. In the second stage results obtained in the first stage are refined via dynamic searching and crawling. This is done by feeding the search results from the first phase to the Mapuccino's dynamic mapping system which augments those results by dynamically crawling in directions where relevant information is found.

Natural language processing has an important role in developing advanced search systems because of the fundamental problem of presenting accurate meaning of search request to the search system. Natural language includes a lot of ambiguities on various levels: morphological, syntactic, semantic, discourse and pragmatic levels, and natural language processing can be used in all stages of information retrieval processing. Some of the research directions in this field are entity extraction, cross language retrieval, automatic information categorization, and summarization. Entity extraction may e.g. enable extraction of names of people, places, and chronological information from text, store them and enable retrieval of this information with natural language search requests. Summarization attempts to automatically reduce document text to its most relevant content based on the user requirements.

An example of research in use of natural language processing on search engines is a system that performs query expansion using an online lexical database WordNet [2]. This system first disambiguates the sense of the query words using the WordNet database, so that each keyword in the query is mapped into its corresponding semantic form as defined in WordNet[5]. After that, WordNet is used for finding synonyms with the query semantic concepts that are subsequently used in Internet search. Documents found in the search are subjected to a new search using the operator that extracts only the paragraphs that provide relevant information to the query. The goal of the system is not to retrieve entire documents but to provide the user with answers, so it returns to the user the paragraphs that may contain the answer.

Software agents (bots) are used for various search activities like searching remote databases, multiple virtual bookstores, or searching and indexing Internet resources. One of the research projects from the MIT Media Lab is the *Expert Finder* project that helps in finding someone who is an expert on some area. The project is based on the assumption that each user has an agent that knows about his or her areas and levels of expertise. When necessary, the user asks the agent to find another user that is an expert on some area. The agent then goes out on the Internet and exchanges information with other expert finder agents, getting their user's profiles. After that, the agent presents to the user a list of people it thought may be able to help, with best candidates first. User can now exchange messages with the selected experts.

Intensive use of the XML markup language is expected to drastically improve information retrieval on the Web and make it more efficient. XML (eXtensible Markup Language)[3] provides a standard system for browsers and other applications to recognize the type of data in documents. This enables search engines to search only certain fields in the documents, instead of the whole documents. This will lead to much faster and more precise search. However, in order to enable use of XML each industry will have to set up its own standards for document structures, Web site providers will have to tag the pages according to the standard structures, and search engines indexing applications will have to hold tag information as metadata.

Currently there are more than 400 million mobile phone users in the world, and it is estimated by Forrester Research that by 2004, 95% of all mobile users will be Internet enabled. Because of this, in the beginning of 2000 several search engines start offering search under the Wireless Application Protocol (WAP). *FAST* search engine thus launched its *WAP search engine* in February 2000, offering search of the index with more than 100,000 WML (Wireless Markup Language) documents, growing at a rate of several hundred percents per month. Besides general search, FAST is preparing search for location information (e.g. location of nearest hospitals or hotels), real-time alerts (e.g. about congested traffic in the area) and image and video streaming. *Google* search engine offers *WAP search* of both WML and HTML documents, since less than 1% of Web sites are available in WML. When a wireless user requests a HTML page, Google translates the requested HTML document on the fly into WML.

1.6 Conclusions

In the last few years significant improvements were made in a number of areas of the World Wide Web search. This session first presented the current state of the World Wide Web, the speed of its development, as well as its heterogeneity. After that, the paper discussed some of the most advanced recent approaches to the Web search, as well as various novel operational Web search elements or systems in the area of subject trees, search engines and database search. There is certainly room for further advancements of both technology and services. One type of service that would be extremely helpful for professionals are *specialty search services* that would exclusively cover the domain of interest of specific groups of professionals. Coverage should be deep, and information should be fresh and almost without any dead links. The objective of this report is to discuss about the specialized search called semantic search, the approaches to implement it, its advantages and disadvantages.

1.7 Report Organization

The second chapter of this report covers the basic knowledge of Semantic web and the WordNet which is useful in designing the semantic based search. The third chapter covers the drawbacks of general key word search and the two approaches – Semantic web search and the Generic relevance search to over come those drawbacks. The fourth and fifth chapters discuss the design and implementation parts of the two semantic search approaches that are mentioned above. The sixth chapter shows some of the results with snap shots and the seventh chapter concludes the report.

Chapter 2

LITERATURE SURVEY

2.1 Semantic Web

2.1.1 Introduction

"The Semantic Web is the representation of data on the World Wide Web. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on the Resource Description Framework (RDF), which integrates a variety of applications using XML for syntax and URIs for naming." – W3C Semantic Web [1]

The concept of the Semantic Web was brought by Tim Berners-Lee who is the inventor of the WWW, URIs, HTTP, and HTML. Today's Web is a human-readable Web where information cannot be easily processed by machine. The efforts of the Semantic Web are to make a machine processable form for expressing information.

Nowadays, there are a huge amount of resources on the Web, which raises a serious problem of accurate search. This is because data in HTML files is useful in some contexts but meaningless under other conditions. In addition, HTML cannot provide description of data encapsulated in it. For example, we want to find an address' details and know its postcode. Since the names of the postcode system are different in many countries and the Web doesn't represent this relationship, we may not get what we expect. By contrast, in the Semantic Web, we can indicate this kind of relationship such as zip code is equivalent to postcode. So when the majority of data on the Web are presented in this form, it is difficult to use such data on a large scale [3]. Another shortcoming is that today's Web lacks an efficient mechanism to share the data when applications are developed independently. Hence, it is necessary to extend the Web to make data machine-understandable and integrated and reusable across various applications [URL1].

To make the Semantic Web work, well-structured data and rules are necessary for agents to roam the Web [URL2]. XML and RDF are two important technologies: we can create our own structures by XML without indicating what they mean; RDF uses sets of triples which express basic concepts. DAML is the extension of XML and RDF.

2.1.2 Introduction to URI - Uniform Resource Identifier

URIs are strings that can identify resources in the Web [URL5]. By using URIs, we can use the same simply naming way to refer to resources under different protocols: HTTP, FTP,
GOPHER, EMAIL etc [URL5]. URLs (Uniform Resource Locator), a widely used type of URIs, are very commonly used in the web, which are addresses of resources. Although often referred to as URLs, URIs can also refer to concepts in the Semantic Web [URL8], e.g. suppose you have a book with the title "Machine Learning", then its URI will look like this: http://www.iitr.ernet.in/home/pw2538/book/title#machinelearning
Here are some other examples of URIs:

uuid:04b749bf-3bb2-4dba-934c-c92c56b709df is a UUID, which stands for a Universal Unique Identifier. UUID can be got by combining the time and the address of your Ethemetcard or a random number, which is then a unique identifier [URL8].

mailto:mysaipec@iitr.ernet.in identifies the mail address of a person.
Note that everything on the Web has a unique URI.

2.1.3 Introduction to HTTP

The Hypertext Transfer Protocol (HTTP) is a generic and stateless protocol for distributed, collaborative, hypermedia information systems. It allows performing operations on resources which are identified by URIs. It has been widely used for more than one decade and now comes to version 1.1. It has four main methods:

a. Get means get the information that is identified by the request URI [URL10]. Usually it is the action we take when browsing sites and clicking hyperlinks -- we ask the web server for the resources that is identified by Request-URLs.

b. Post means make a request to the web server so that the server accepts the resources encapsulated in the request, which will be the new subordinate of the resource identified by the Request-URI in the Request-Line [URL10]. Usually it is what we do when filling and sending the HTML form to the server to buy something like CDs, books etc. -- making a request of the server.

c. Put means make a request that sends updated information about a resource if the resource identified by the Request-URI exists, otherwise the URI will be regarded as a new resource [URL10]. The main difference between the POST and PUT requests lies in the different meaning of the Request-URI. In a POST request, the URI is to identify the resource that will handle the enclosed entity. As for the PUT request, the user agent knows what URI is its aim and the web server cannot redirect the request to other resources. Unfortunately most web browsers don't implement this functionality, which makes the Web, to some extent, a one-way medium [URL8].

d. Head is similar to GET except that the servers don't return a message-body in the response. The benefit of this method is that we can get meta-information about the entity implied by the request without transferring the entity-body itself. We can use this method to check if hypertext links are valid, or if the content is modified recently [URL10].

By using HTTP, Semantic Web can benefit all these functionalities for free. In addition, almost all HTTP servers and clients support all these features.

2.1.4 XML

Extensible Markup Language (XML) is a subset of SGML (the Standard Generalized Markup Language) [URL9], i.e. it is totally compatible with SGML. But it is simple and flexible. It's original aim to tackle the problems of large-scale electronic publishing. However, it is also very important in data exchange on the Web. Despite its name, XML is not a markup language but a set of rules to build markup languages.

2.1.4.1 Markup language

"Markup is information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how they relate to each other." - Erik T. Ray, Page 10 [17]. Markup language is kind of mechanism organizing the document with a set of symbols, e.g. this article is labeled with different fonts for headings. Markup use similar methods to achieve its aims. Markup is important to implement machine-readable documents since a program need to treat different part of a document individually.

2.1.4.2 Why XML?

HTML cannot provide arbitrary structure and it is bound with a set of semantics [18], which result in weak flexibility. By contrast, XML is a meta-language which can build markup languages. XML itself does not specify preconceived semantics and predefined tag sets [18], so the semantics of XML will be defined by other applications. As for SGML, it is too complicate to be implemented in web browsers although it can do everything XML can do.

2.1.4.3 XML Documents

XML documents are similar to HTML documents, bound with some tags. For example:

```
<?xml version='1.0'?>
<!-- This file represents a fragment of a book store inventory database -->
<bookstore>
        <book genre="science">
                <title>Machine Learning</title>
                <author>
                        <first-name>Tim</first-name>
                        <last-name>Mitchell</last-name>
                </author>
                <price>28.99</price>
        </book>
        <book genre="travel">
                <title> Family Fun Vacation Guides </title>
                <author>
                        <first-name>Jill</first-name>
                        <last-name>Mross</last-name>
                </author>
                <price>12.57</price>
        </book>
</bookstore>
```

Fig: 2.1 An example of XML document

A XML document is composed of pieces called **elements** which are the most common form of markup. Elements are always enclosed with a start-tag, *<element>* and an end-tag *</element>* if it is not empty.

Attributes are associate name-value pairs which lie in the elements. For example, *<book genre="philosophy">* is a *<book>* element where the genre attribute has the value philosophy. Attributes must be quoted with single or double quotes in XML documents.

2.1.4.4 Namespaces

We can expand our vocabulary by namespaces which are groups of element and attribute names. Suppose, if you want to include a symbol encoded in another markup language in an XML document, you can declare the namespace that the symbol belongs to. In addition, we can avoid the situation that two XML objects in different namespaces with the same name have different meaning by the feature of namespaces [17]. The solution is to assign a prefix that indicates which namespace each element or attribute comes from [17]. The syntax is shown below:

ns-prefix:local-name

2.1.4.5 XML Schemas

XML itself does not do anything, i.e., it is just structure and store information. But if we need a program to process the XML document, there must be some constraints on sequence of tags, nesting of tags, required elements and attributes, data types for elements and attributes, default and fix values for elements and attributes and so on. XML Schema is an XML based alternative to Document Type Definition (DTD) [URL11]. There are some features of XML Schemas that overweigh DTD:

a. XML Schemas support data types, which brings a lot of benefits, e.g. easy to validate the correctness of data, easy to work with databases, easy to convert data between different types.
b. XML Schemas have the same syntax as XML so that it can benefit all features of XML. XML Schemas secure data communication since it can describe the data in machine-understandable way.
c. XML Schemas are extensible because they are actually XML and then share this feature of XML.

d. Well-formed is not enough since it also may contain some semantic confusion which can be caught by XML Schemas.

2.1.4.6 Well-Formed and Valid Documents

An XML document is well-formed only if it meets all following requirements:

a. There is one, and only one, root element [4].
b. Each tag must be closed [4].
c. "Tag names are case-sensitive" [4].

A well-formed XML document is valid only if it refers to a proper DTD or XML Schema so that the document obeys the constraints of that DTD ~~or XML Schema. [4].~~

2.1.5 RDF

The following subsections provide detailed information about resource description framework.

2.1.5.1 Metadata

Metadata is information about information [URL12], which is widely used in real world for searching. For example, you want to borrow some books on computer from a library. Usually a library will provide a lookup system which allows you to list books by author, title, subject and so on. This list contains lots of useful information: author, title, ISBN, date and most important, location of the book. You need some information (the book's location) you want to know and you use metadata (information about information, in this case: author, title and subject) to get it. However, metadata is not necessary [URL12]: you can lookup the book you want to find one by one among all books in the library. Obviously this is not a wise way. In addition, the use of metadata is not just for searching although searching is the most common aim of metadata. There is some other useful information behind the scenes, which are important to business.

2.1.5.2 What is RDF?

Resource Description Framework is a framework for processing metadata [URL12] and it describes relationships among resources with properties and values [URL15]. It is built on the following rules:

a. **Resource**: Everything described by RDF expressions is called a resource [URL14]. Every resource has a URI and it may be an entire web page or a part of a web page [URL12, URL14]

b. **Property**: "A property is a specific aspect, characteristic, attribute, or relation used to describe a resource" – W3C, Resource Description Framework (RDF) Model and Syntax Specification [URL14]. Note that a property is also a resource since it can have its own properties.

c. **Statements**: A statement combines a resource, a property and a value [URL14]. These three individual parts are known as the "subject", "predicate" and "object". For example, "The Author of http://www.iitr.ernet.in/home/pw2538/index.html is Allis" is a statement. Note that value can be either a string or another resource.

2.1.5.3 Examples

Statements can be represented as a graph in RDF. First consider a simple example:

Krishna is the author of the resource http://www.cs.iitr.ernet.in/home/pw2538/index.html

This sentence has the following parts:

Subject (Resource)	http://www.cs.iitr.ernet.in/home/pw2538/index.html
Predicate (Property)	Author
Object (literal)	"Krishna"

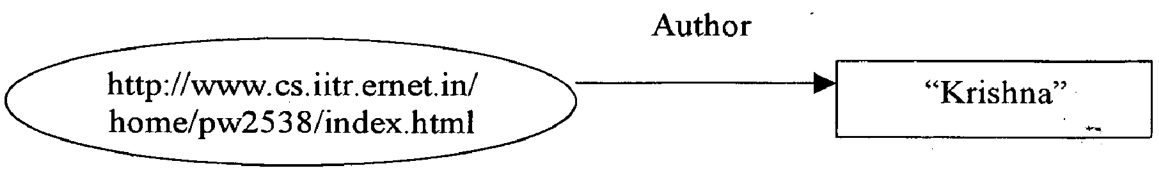


Fig: 2.2 Simple node and arc diagram

The direction of the arrow is always from the subject to the object of the statement. And the graph can be read in the way: "<subject> HAS <predicate> <object>" [URL14], i.e. "http://www.cs.iitr.ernet.in/home/pw2538/index.html has the author Krishna".

If we assign a URI to the *author* property:

http://www.cs.iitr.ernet.in/home/pw2538/terms/author

In order to represent briefly, we make some prefixes to avoid writing URI references completely. There are some well-known QName prefixes;

prefix rdf:, namespace URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#

prefix rdfs:, namespace URI: http://www.w3.org/2000/01/rdf-schema#

prefix daml:, namespace URI: http://www.daml.org/2001/03/daml+oil#

prefix xsd:, namespace URI: http://www.w3.org/2001/XMLSchema#

Here we use a prefix pwterms to represent our own URI references

Prefix pwterms:, namespace URI: http://www.cs.iitr.ernet.in/home/pw2538/terms

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:pwterms=" http://www.cs.iitr.ernet.in/home/pw2538/terms">
<rdf:Description rdf:about="http://www.cs.iitr.ernet.in/home/pw2538/index.html">
        <pwterms:author>Krishna</pwterms:author>
</rdf:Description>

</rdf:RDF>
```

Fig: 2.3 RDF for a Simple RDF Statement

Now consider a more complicate example:

The individual referred to by student id pw2538 is named Krishna and has the email address krishpec@iitr.ernet.in This individual is the author of the resource http://www.cs.iitr.ernet.in/home/pw2538/index.html.

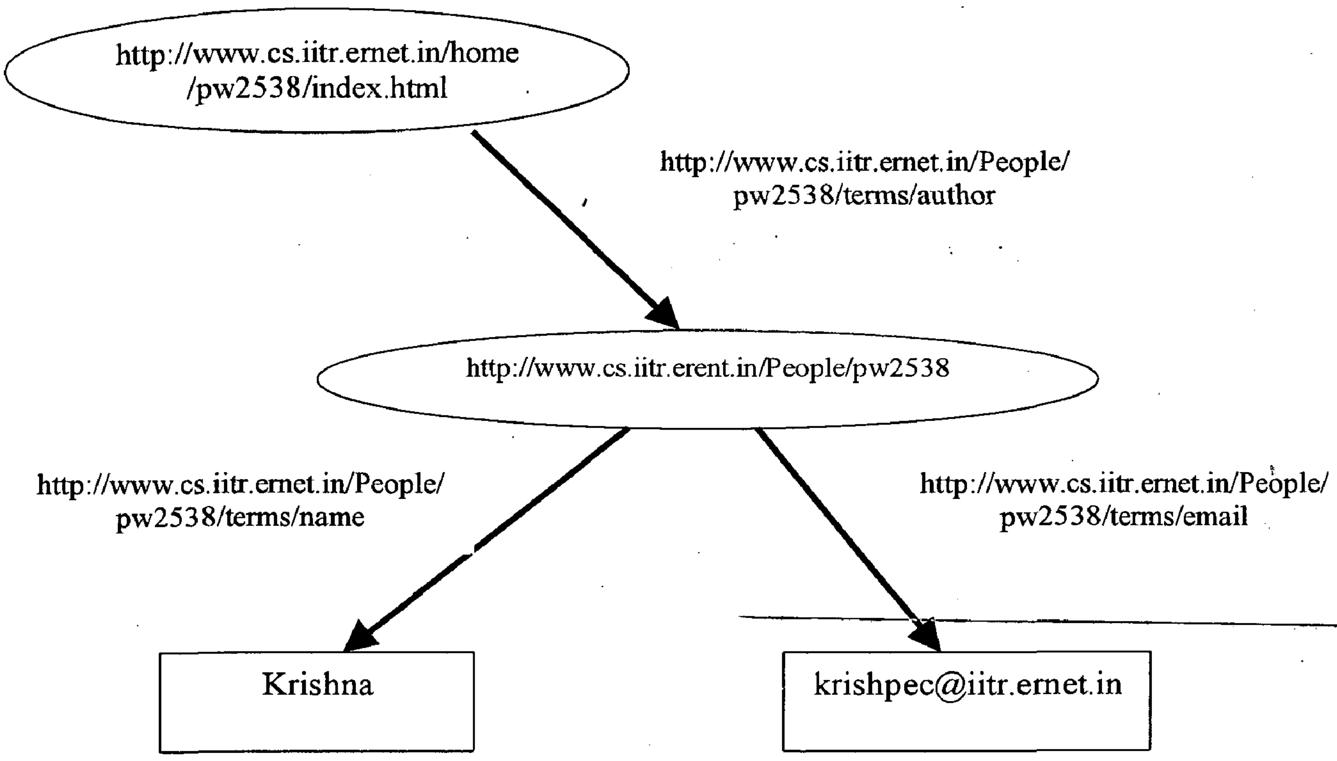

http://www.cs.iitr.ernet.in/home /pw2538/index.html
http://www.cs.iitr.ernet.in/People/ pw2538/terms/author
http://www.cs.iitr.erent.in/People/pw2538
http://www.cs.iitr.ernet.in/People/ pw2538/terms/name
http://www.cs.iitr.ernet.in/People/ pw2538/terms/email
Krishna
krishpec@iitr.ernet.in

Fig: 2.4 Structured value with identifier

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:pwterms=" http://www.cs.iitr.ernet.in/home/pw2538/terms">

<rdf:Description about="http://www.cs.iitr.erent.in/home/pw2538/index.html">
        < pwterms:author rdf:resource="http://www.cs.iitr.erent.in/People/pw2538"/>
</rdf:Description>

<rdf:Description about="http://www.cs.iitr.erent.in/People/pw2538">
        <pwterms:Name>Krishna</pwterms:Name>
         <pwterms:Email>krishpec@iitr.ernet.in</pwterms:Email>
</rdf:Description>

</rdf:RDF>
```

Fig: 2.5 RDF for a complicate RDF Statement

2.1.5.4 Why Not Just Use XML?

Since RDF is based on XML and XML can also represent the statements in a natural way, why not just use XML instead of using a new language RDF [URL12]. However, XML has some shortcoming when dealing with metadata:

a. In XML documents, the order of elements is often very important and meaningful. However, in metadata, this is redundant; for instance, we don't care whether a book is listed first when we look up in the library. Furthermore, it will reduce the performance and efficiency if maintaining the correct order of data items [URL12].
b. XML allows mixed structures like

```
<Description>
Here, XML allows mixture of text and child properties; for example, its width
(<Width>30</ Width >) and height (<Height>20</Height>).
</Description>
```

Fig: 2.6 Partial XML document with mixture structure

So data structures in XML will include the mixture of trees, graphs, and character strings [URL12]. In general, it requires more computation when dealing with these complicate structures. By contrast, RDF is more straightforward.

2.1.5.5 RDF Schemas

Although RDF can easily describe resources, we still need a mechanism to figure out what a specific term means and how it should be used [URL3]. This is the function of the RDF vocabulary description language, RDF Schema. RDF Schema is a simple data-typing model for RDF [URL3] so that we can describe groups of related resources and the relationships among these resources [URL15]. For example, we can say "pupil" is a type of "student" and "student" is a subclass of "people".

Resources can be divided into "classes" which is composed of instances [URL15]. A class itself is also a resource which is usually identified by RDF URI References and can be described by RDF properties. We often use the prefix "rdfs:" to indicate the term is RDF Schema term. “rdfs:Resource” is the root class of everything in RDF Schema. “rdf:type” is an instance of rdf:Property (class of RDF properties), and it means that a resource is an instance of a class. The property rdfs:subClassOf is an instance of rdf:Property that is used to state a class is a subclass of the other. Figure 2.7 shows that “Animal” is the super-class in this RDF document, i.e. “Dog”, “Cat”, and “PersianCat” are all subclasses of “Animal”.

```
<?xml version="1.0"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdf:Description rdf:ID="Animal">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="Dog">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:subClassOf rdf:resource="#Animal"/>
</rdf:Description>

<rdf:Description rdf:ID="Cat">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:subClassOf rdf:resource="#Animal"/>
</rdf:Description>
<rdf:Description rdf:ID="PersianCat">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:subClassOf rdf:resource="#Cat"/>
</rdf:Description>
</rdf:RDF>
```

Fig: 2.7 The Animal Class Hierarchy in RDF/XML

Although "PersianCat" class doesn't directly indicate this relation, the relationship can be got from: "PersianCat" is the subclass of "Cat" and "Cat" is the subclass of "Animal", then "PersianCat" is also the subclass of "Animal". This is similar to the feature inheritance in the Object-oriented programming theory.

2.1.6 DAML

The DARPA Agent Markup Language (DAML) Program started in 2000. DAML combines many language components of the Ontology Inference Layer (OIL) soon after it was started. The result of these efforts is DAML+OIL, a more robust language for general knowledge representation than RDF and RDFS. DAML is not a W3C standard, but many people in W3C participated in this program. DAML is kind of extension of RDF and RDFS, but it is not a data model. It not only provides stronger abilities to express constraints in schemas but also can build general knowledge representation, i.e. it is also an ontology language.

2.1.6.1 Introduction to DAML

DAML extends RDF and RDFS by adding more support for data typing and semantics. These improvements lie in the enhancement of properties and classes.

a. **Properties**: DAML add a primitive "*DatatypeProperty*" that allows strict data types that defined in XML Schemas or user-defined data types e.g. float number, integer and so on. In DAML, a property can have multiple ranges, which brings rich flexibilities. Furthermore, DAML allows we declare a unique property, i.e. there are no two instances with same value. This is the function of a primitive "*daml:UniqueProperty*". We can also describe the relation between two properties that are equivalent by either "*daml:samePropertyAs*" or "*daml:equivalentTo*". In addition, there are more powerful features in properties in DAML, with which we can express relations such as "inverse", "transitivity". If A is the employer of B, then B is the employee of A. The properties "employer" and "employee" are the inverse of each other. This relation can be expressed with "*daml:inverseOf*'. The "transitivity" means that if A is a subset of B, and B is a subset of C, then A must be a subset of C. The property "*daml:TransitiveProperty*" is used to express this relation. More interestingly, DAML provides "*daml:onProperty*", "*daml:hasValue*", "*daml:hasClass*" and "*daml:toClass*" to restrict classes to a set of resources based on particular properties. Then we can make the rules for a specific class so that a resource can be a member of the class if and only if its properties must satisfy the requirements. "*daml:onProperty*" identifies the property to be checked. We can define property restrictions by its value with "*daml:hasValue*", i.e. the property must have a particular value. "*daml:hasClass*" can be used to define property restrictions by the class of the values of a property, instead of its value. By contrast, "*daml:toClass*" is more restrictive since it requires that **all** the property values for a resource must be a particular class. However, a resource without property given "*daml:onProperty*" can also satisfy the condition. So this feature must be used very carefully.

b. **Classes**: "*daml:Class*" is a subclass of "*rdfs:Class*" and DAML adds many wonderful features in it. We can build more expressiveness description of resources with these features. We can define an enumeration that cannot be implemented in RDF. In DAML, "*daml:oneOf*" element defines an enumeration. We can also define a closed list by declaring "*daml:oneOf*" to be the "*daml:collection*" parse type. Additionally, we can build some relations such as "disjoint", "union" and "intersection". Both "*daml:disjointWith*" and "*daml:disjointUnionOf*" can be used to assert there are no instances in common among classes. Non-exclusive boolean combinations of classes can be expressed with "*daml:unionOf*". The "*daml:intersectionOf*" property can express intersection of the sets.

2.1.6.2 Why DAML?

RDF is very straightforward to implement, which is both its advantage and disadvantage. It is not enough when we want more strict data typing and a consistent expression for enumerations and so on. For example, we want to describe a book sold by Amazon. Below is the RDF and RDFS form.

```
<rdfs:Class rdf:ID="Book">
        <rdfs:label>Book</rdfs:label>
        <rdfs:comment>A book sold by Amazon</rdfs:comment>
</rdfs:Class>

<rdfs:Property rdf:ID="pages">
        <rdfs:label>Pages</rdfs:label>
        <rdfs:domain rdf:resource="#Book"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdfs:Property>

<Book rdf:ID="MachineLearning">
        <rdfs:label>Machine Learning</rdfs:label>
        <pages>432</pages>
</Book>
```

Fig: 2.8 Book Example with RDF and RDFS

The disadvantage of the above form is that literals can be any string, but we expect that pages must be a positive integer. Compared with RDF and RDFS, DAML allow us to use a more accurate data type (defined in XSD) to describe data. Apart from these advantages, there are many DAML data sets open to public on the Web.

```
<daml:DatatypeProperty rdf:ID="pages">
        <rdfs:label>Pages</rdfs:label>
        <rdfs:domain rdf:resource="#Book"/>
        <rdfs:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#positiveInteger"/>
</daml:DatatypeProperty>
```

Fig: 2.9 Book Example with DAML

2.2 About WordNet

Standard alphabetical procedures for organizing lexical information put together words that are spelled alike and scatter words with similar or related meanings haphazardly through the list. Unfortunately, there is no obvious alternative, no other simple way for lexicographers to keep track of what has been done or for readers to find the word they are looking for. In this age of computers, however, there is an answer to that complaint. One obvious reason to resort to on-line dictionaries—lexical databases that can be read by computers—is that computers can search such alphabetical lists much faster than people can. A dictionary entry can be available as soon as the target word is selected or typed into the keyboard. Once computers are enlisted in the service of dictionary users, however, it quickly becomes apparent that it is grossly inefficient to use these powerful machines as little more than rapid page-turners. The challenge is to think what further use to make of them. WordNet[5] is a proposal for a more effective combination of traditional lexicographic information and modern high-speed computation.

WordNet is like a dictionary in that it stores words and meanings. However it differs from traditional ones in many ways. For instance, words in WordNet are arranged *semantically* instead of alphabetically. Synonymous words are grouped together to form synonym sets, or *synsets*. Each such synset therefore represents a single distinct sense or *concept*. Thus, the synset {base, alkali} represents the sense of any of various water-soluble compounds capable of turning litmus blue and reacting with an acid to form a salt and water.

Words with multiple senses can either be *homonymous* or *polysemous*. Two senses of a word are said to be homonyms when they mean entirely different things but have the same spelling. For example the two senses of the word *bark* – tough protective covering of trees and the sound made by a dog are homonyms because they are not related to each other. A word is said to be polysemous when its senses are various shades of the same basic meaning. For example, the word *accident* is polysemous since its two senses – a mishap and anything that happens by chance are somewhat related to each other. Note that WordNet does not distinguish between homonymous and polysemous words, and therefore neither do we. Thus WordNet does not indicate that the two senses of the word *accident* are somewhat closer to each other in meaning than the two senses of the word *bark*.

Words with only one sense are said to be *monosemous*. For example, the word *wristwatch* has only one sense and therefore appears in only one synset. In WordNet, each word occurs in as many synsets as it has senses. For example the word *base* occurs in two noun synsets, {base, alkali} and {basis, base, foundation, fundament, groundwork, cornerstone}, and the verb synset {establish, base, ground, found}.

WordNet stores information about words that belong to four parts–of–speech: nouns, verbs, adjectives and adverbs. Besides single words, WordNet synsets also sometimes contain *compound words* which are made up of two or more words but are treated like single words in all respects. Thus for example WordNet has two–word compounds like *banking concern* and *banking company*, three–word compounds like *depository financial institution*, four–word compounds like *keep one's eyes peeled* etc.

Each synset in WordNet has an associated definition or *gloss*. This consists of a short entry explaining the meaning of the concept represented by the synset. Many (but not all) synsets also contain example sentences that show how the words in the synset may be used in English.

WordNet defines a variety of *semantic* and *lexical* relations between words and synsets. Semantic relations define a relationship between two synsets. Lexical relations on the other hand define a relationship between two words within two synsets of WordNet. Thus whereas a semantic relation between two synsets relates all the words in one of the synsets to all the words in the other synset, a lexical relationship exists only between particular words of two synsets. Synonymy and antonymy form the lexical relations whereas hyponymy/hypernymy (variously called subordination/superordination, subset/superset, or the ISA relation) and meronymy/holonymy (is the part-whole or HASA relation) form semantic relations.

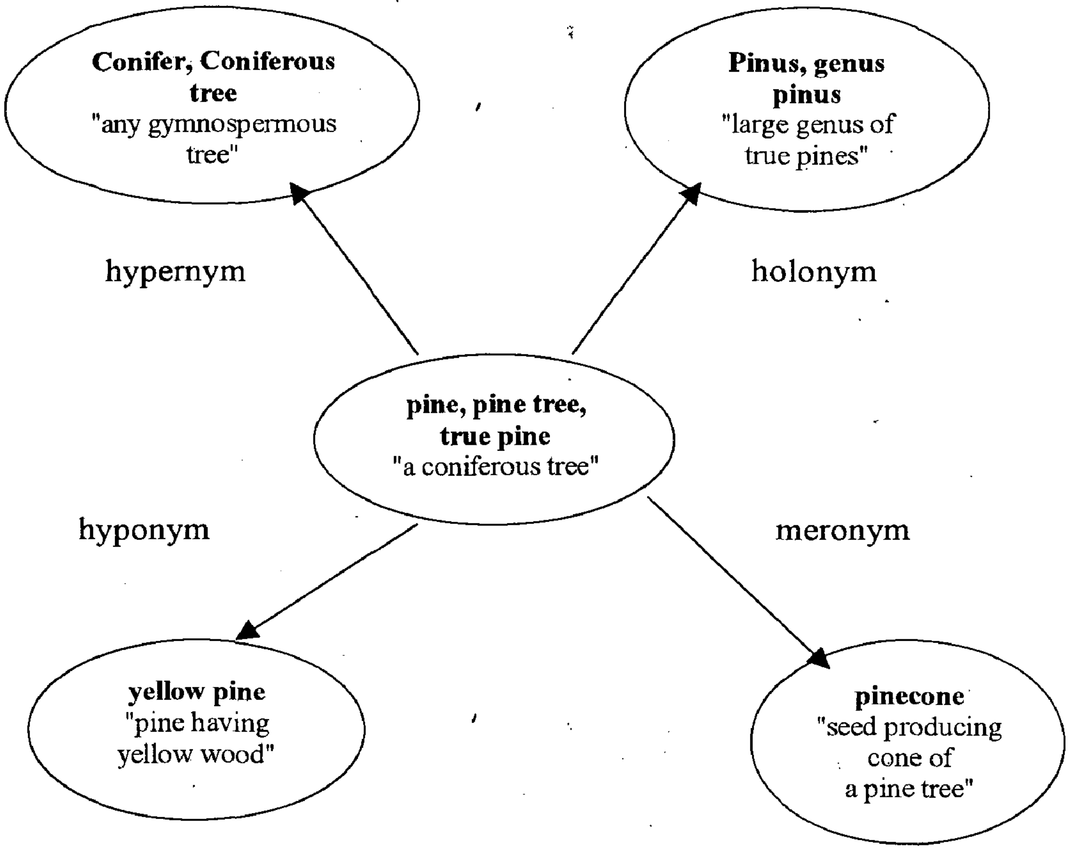


Fig:2.10 Relations in WordNet

2.2.1 WordNet Database

For each syntactic category, two files are needed to represent the contents of the WordNet database – **index.***pos* and **data.***pos*, where *pos* is **noun**, **verb**, **adj** and **adv**. The other auxiliary files are used by the WordNet library's searching functions and are needed to run the various WordNet browsers. Default file names for the WordNet database are index.noun, data.noun, index.verb, data.verb, index.adj, data.adj, index.adv, data.adv

Each index file is an alphabetized list of all the words found in WordNet in the corresponding part of speech. On each line, following the word, is a list of byte offsets (*synset_offsets*) in the corresponding data file, one for each synset containing the word. Words in the index file are in lower case only, regardless of how they were entered in the lexicographer files. This folds various orthographic representations of the word into one line enabling database searches to be case insensitive.

A data file for a syntactic category contains information corresponding to the synsets that were specified in the lexicographer files, with relational pointers resolved to *synset_offsets*. Each line corresponds to a synset. Pointers are followed and hierarchies traversed by moving from one synset to another via the *synset_offsets*.

2.2.1.1 Index File Format

Each index file begins with several lines containing a copyright notice, version number and license agreement. These lines all begin with two spaces and the line number so they do not interfere with the binary search algorithm that is used to look up entries in the index files. All other lines are in the following format. In the field descriptions, number always refers to a decimal integer unless otherwise defined.

lemma pos synset_cnt p_cnt [ptr_symbol...] sense_cnt tagsense_cnt synset_offset [synset_offset...]

lemma	lower case ASCII text of word or collocation. Collocations are formed by joining individual words with an underscore (_) character.
pos	Syntactic category: **n** for noun files, **v** for verb files, **a** for adjective files, **r** for adverb files.

All remaining fields are with respect to senses of *lemma* in *pos*.

synset_cnt	Number of synsets that *lemma* is in. This is the number of senses of the word in WordNet. See Sense Numbers below for a discussion of how sense numbers are assigned and the order of *synset_offsets* in the index files.
p_cnt	Number of different pointers that *lemma* has in all synsets containing it.
ptr_symbol	A space separated list of *p_cnt* different types of pointers that *lemma* has in all synsets containing it. If all senses of *lemma* have no pointers, this field is omitted and *p_cnt* is **0**.
sense_cnt	Same as *sense_cnt* above. This is redundant, but the field was preserved for compatibility reasons.
tagsense_cnt	Number of senses of *lemma* that are ranked according to their frequency of occurrence in semantic concordance texts.
synset_offset	Byte offset in **data.***pos* file of a synset containing *lemma*. Each *synset_offset* in the list corresponds to a different sense of *lemma* in WordNet. *synset_offset* is an 8 digit, zero-filled decimal integer that can be used to read a synset from the data file.

2.2.1.2 Data File Format

Each data file begins with several lines containing a copyright notice, version number and license agreement. These lines all begin with two spaces and the line number. All other lines are in the following format. Integer fields are of fixed length, and are zero-filled.

synset_offset lex_filenum ss_type w_cnt word lex_id [word lex_id...] p_cnt [ptr...] [frames...] |gloss

synset_offset	Current byte offset in the file represented as an 8 digit decimal integer.
lex_filenum	Two digit decimal integer corresponding to the lexicographer file name containing the synset.
ss_type	One character code indicating the synset type: **n** NOUN **v** VERB **a** ADJECTIVE **s** ADJECTIVE SATELLITE **r** ADVERB
w_cnt	Two digit hexadecimal integer indicating the number of words in the synset.
word	ASCII form of a word as entered in the synset by the lexicographer, with spaces replaced by underscore characters (_). The text of the word is case sensitive, in contrast to its form in the corresponding **index.***pos* file, that contains only lower-case forms. In **data.adj**, a *word* is followed by a syntactic marker if one was specified in the lexicographer file. A syntactic marker is appended, in parentheses, onto *word* without any intervening spaces.
lex_id	One digit hexadecimal integer that, when appended onto *lemma*, uniquely identifies a sense within a lexicographer file. *lex_id* numbers usually start with **0**, and are incremented as additional senses of the word are added to the same file, although there is no requirement that the numbers be consecutive or begin with **0**. Note that a value of **0** is the default, and therefore is not present in lexicographer files.

p_cnt	Three digit decimal integer indicating the number of pointers from this synset to other synsets. If *p_cnt* is **000** the synset has no pointers.
Ptr	A pointer from this synset to another. *ptr* is of the form:

pointer_symbol synset_offset pos source/target

where synset_offset is the byte offset of the target synset in the data file corresponding to pos.

The source/target field distinguishes lexical and semantic pointers. It is a four byte field, containing two two-digit hexadecimal integers. The first two digits indicates the word number in the current (source) synset, the last two digits indicate the word number in the target synset. A value of **0000** means that pointer_symbol represents a semantic relation between the current (source) synset and the target synset indicated by *synset_offset*. A lexical relation between two words in different synsets is represented by non-zero values in the source and target word numbers. The first and last two bytes of this field indicate the word numbers in the source and target synsets, respectively, between which the relation holds. Word numbers are assigned to the word fields in a synset, from left to right, beginning with **1**.

Frames	In **data.verb** only, a list of numbers corresponding to the generic verb sentence frames for *word*s in the synset. *frames* is of the form:

f_cnt + f_num w_num [+ f_num w_num...]

where *f_cnt* a two digit decimal integer indicating the number of generic frames listed, *f_num* is a two digit decimal integer frame number, and *w_num* is a two digit hexadecimal integer indicating the word in the synset that the frame applies to. As with pointers, if this number is **00**, *f_num* applies to all *word*s in the synset. If non-zero, it is applicable only to the word indicated. Word numbers are assigned as described for pointers. Each *f_num w_num* pair is preceded by a +. See **wninput**(5WN) for the text of the generic sentence frames.

gloss	Each synset contains a gloss. A *gloss* is represented as a vertical bar (|), followed by a text string that continues until the end of the line. The gloss may contain a definition, one or more example sentences, or both.

2.2.1.3 Sense Numbers

Senses in WordNet are generally ordered from most to least frequently used, with the most common sense numbered **1**. Frequency of use is determined by the number of times a sense is tagged in the various semantic concordance texts. Senses that are not semantically tagged follow the ordered senses. The *tagsense_cnt* field for each entry in the **index.***pos* files indicates how many of the senses in the list have been tagged.

When the **index.***pos* files are generated, the *synset_offset*s are output in sense number order, with sense 1 first in the list. Senses with the same number of semantic tags are assigned unique but consecutive sense numbers. The WordNet **OVERVIEW** search displays all senses of the specified word, in all syntactic categories, and indicates which of the senses are represented in the semantically tagged texts.

Chapter 3

ANALYSIS

3.1 General Search Engine Functionalities

There are differences in the ways various search engines work, but they all perform three basic tasks:

- They search the Internet or select pieces of the Internet, based on important words.
- They keep an index of the words they find, and where they find them.
- They allow users to look for words or combinations of words found in that index.

This chapter will discuss the major drawbacks of keyword search and the semantic solutions to those problems.

3.2 Problems with General Key Word Search

On the whole the task of information search on key words is solved rather well, and in combination with AND/OR/NOT-inquiries satisfies almost all requirements of web-users. However, life and market dictate their own rules, urging the developers to seek for more effective ways to meet the user's requirements in both search time and quality of information selection, thus attracting them to Web sites. It should be emphasized that an experienced user is able to find any information in the web using the disposable tools. Unfortunately, most of the users don't have the necessary qualifications to formulate their information demands; therefore the problem of intellectualization of search engines is vital.

The people professionally involved in natural language processing are aware of the two wide-spread 'drawbacks' of search engines:

- Search irrelevance, or information racket;
- Incompleteness of the search.

For the problem of irrelevancy one of the reason is polysemy. Polysemy is understood here as the situation when several objects, notions and relations are referred to by one and the same word (word combination) In automatic processing polysemy and especially metaphority of the natural language are underestimated. But the problem of polysemy solution appears dominant for machine translation systems, NL-processing systems and organization of qualitative search.

The problem of incompleteness of search can be defined as follows: there is information, which the user is interested in, but it has an indirect connection with the inquiry, so the search engine cannot find it. The most wide-spread reason for the occurrence is synonymy. But synonymic relations do not exhaust the variety of semantic relations in the subject domain.

The cardinal solution of the problem may appear as the development of a gigantic database of encyclopedic and common knowledge to satisfy all information requirements. However, this way is unacceptable at present due to a number of unsolved fundamental scientific problems in artificial intelligence

3.3 Solutions to Keyword Search Problems

To overcome the problems of keyword search two approaches that can be adopted are:

- Search based on Semantic Web
- Search based on Generic Relevance

3.3.1 Semantic Web Search

The Semantic Web comprises recent work to improve the relevance of search results by integrating semantics into Web pages. This work focuses primarily on defining standards based on an ontology or a metadata model and syntax like the World Wide Web Consortium's resource description framework (http:// www.w3.org/RDF/). Semantic frameworks are arguably an interesting approach for a single organization or a limited topic, especially where particular functionality exists to restrict the ontology and thus allow semiautomatic support for query processing. The approach has also proven useful for navigating subject-specific sites.

3.3.1.1 Ontology

The word "ontology" is borrowed from philosophy. Its original meaning is "the branch of metaphysics that deals with the nature of being" -- The American Heritage®

Dictionary of the English Language: Fourth Edition (2000). In AI domain, T. R. Gruber defined the term as a specification of a conceptualization, i.e. it is a description of concepts and relationships with a set of representational vocabulary. The aim of building ontologies is to share and reuse knowledge.

3.3.1.2 Ontology Matching

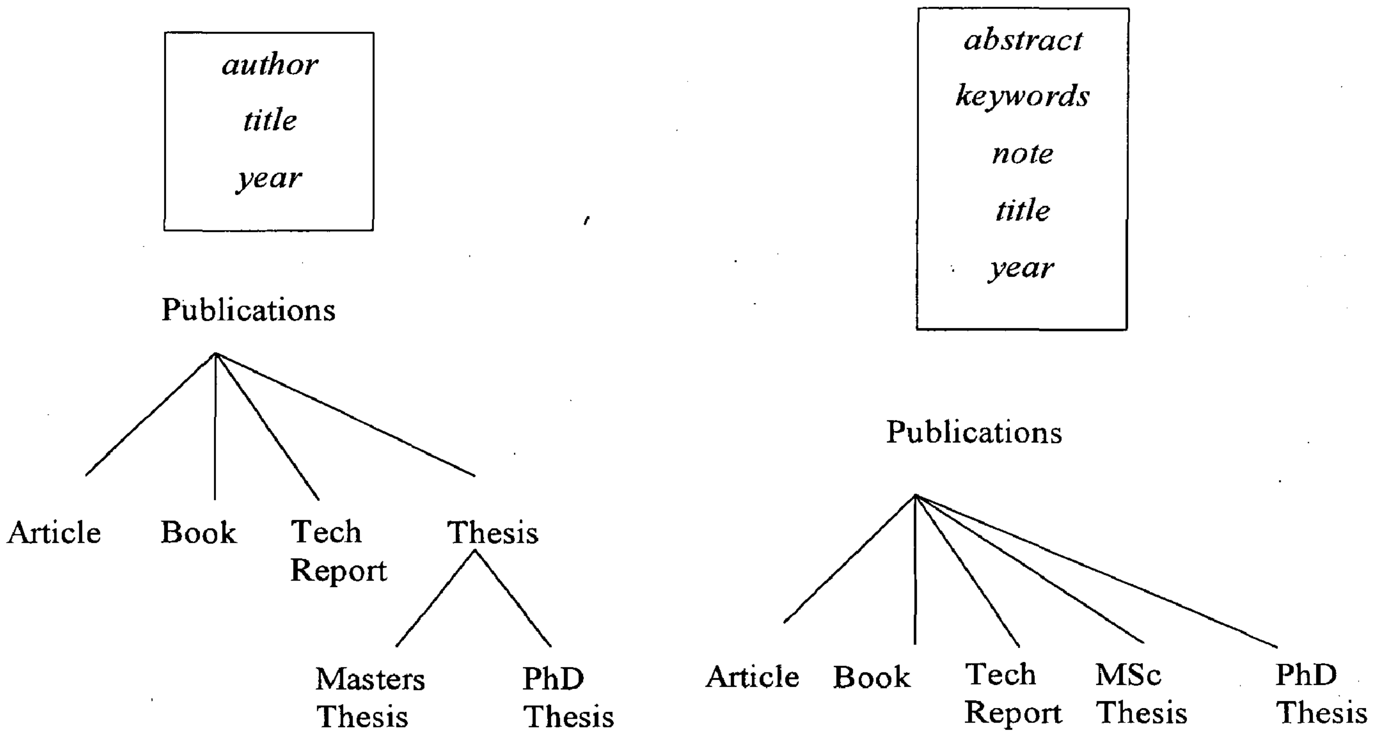


Fig: 3.1 Two Publication Ontologies

Since the Semantic Web is built distributively, there are many different ontologies that describe the semantically equivalent things. Therefore it is necessary to map among elements of these ontologies if we want to process information in the Web scale. An ontology can be represented in taxonomy tree form where each node represents a concept with its attributes. The above figure shows two different publication ontologies. For example, the concept publication on the left of the above figure has three attributes: author, title and year. The aim of ontology matching is to map the semantically equivalent elements. For example, "MastersThesis" maps to "MScThesis" in the above figure. This is a one-to-one mapping , the simplest type. We can also map the different types of elements, e.g. a particular relation maps to a particular attribute. Mapping can be more complex if we want to map the combination of some elements to a specific element. For example, "FullName" maps to the combination of "FirstName" and "LastName".

3.3.2 Generic Relevance Search

Today's Internet search engines work at a lexical level, performing string pattern matching to stored documents and augmenting the results with link analysis, frequency counting, and elementary structural analysis such as weighted title words. These engines thus provide first-level information filtering. The user, however, must still perform most of the relevance filtering.

Relevance is typically person dependent, so personalization becomes important feature of this search. User constructs a profile based on his search intention. Using his profile the search is expanded or narrowed down to a particular topic. This approach is very useful for some restricted domains. To find the user relevant meaning of the given query terms WordNet is used.

One of the *basic mechanisms for lexical polysemy solution,* is to find the Semantics of the given query. *The semantic mechanism* includes paraphrasing of the inquiry with the use of a list of lexical meanings of the entered key words. *The lexical meaning* is introduced in the form of a word, or a word combination, which describes the category of concepts of the meaning.

The mechanisms for the completeness of search:

- Cognitive morphology
- Search on the list of words with the same root morpheme;
- Synonyms search;
- Ontology search;

The inquiry expansion by means of *cognitive morphology* is activated in the cases when the introduced key word lacks in the text data bank. In this case the user is offered a similar word received by revealing of potential spelling mistakes.

The search on the list of words with the same root morpheme is done on the list of words in the WordNet. The user is offered a list of words with the same root morpheme as in the key word.

The synonyms search is done on the list of words in the WordNet. The user is offered a list of words with the synonyms to the key word.

The ontology search includes a list of words closely related to the key word. The ontology search activates the semantic relations, which are latent in word combinations.

Besides, the above mentioned means of polysemy solution can at the same time act as a means for the completeness of search, if a multiple choice from the offered lists and the choice of all meanings are provided.

Chapter 4

DESIGN

4.1 Search Based on Semantic Web

When reviewing ontologies interesting tasks are finding synonyms, homonyms and isomorphism. Synonyms are different words, which have the same meaning. In the case of RDF data they can be seen as different resources used with the same meaning. Homonyms are usages of the same word, in which they have a different meaning. In the case of RDF data they can be seen as a resources used with different meanings. Within an ontology the meaning of a resource is defined by its relations to other resources. Looking at the RDF graph those relations are adjacencies to other nodes.

Finding an isomorphism between to graphs means determining that they have the same structure. Assuming that in an ontology the meaning of a resource is defined by its relations to other resources finding synonyms means finding occurrences of different resources with the same relations to other resources. This is obviously related to finding isomorphism.

4.1.1 Graph Isomorphism

An isomorphism between two graphs g1 and g2 is a bijection f between the nodes in g1 and the nodes in g2 such that for every pair of nodes ni, nj from g1 f(ni) and f(nj) are adjacent iff ni and nj are adjacent.

The problem of finding isomorphism between graphs unfortunately is in NP. There can't be an algorithm, which answers the question whether there is an isomorphism in time (number of calculation steps) related by a polynomial function to the size of the input for the general case.

Consider the two example graphs in Fig.4.1 Since they have a different number of nodes there can't be a complete bijective mapping but a partial one. This is called subgraph isomorphism.

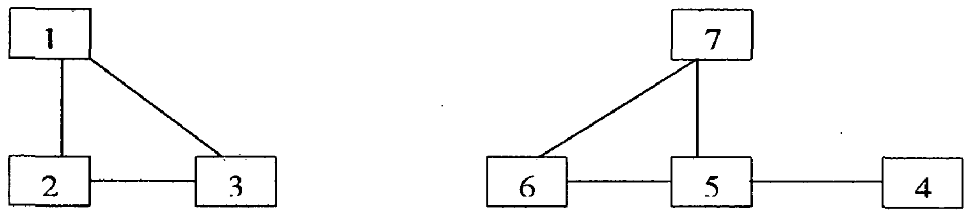


Fig: 4.1 Two Graphs

The two graphs contain very few nodes. But the search space is not that small. All possible maps can be arranged in the tree shown in Fig. 4.2.

Due to the size of the search space it is important to find ways to restrict it. The following section will introduce two basic algorithms for finding graph isomorphism. The later section will discuss how the search for isomorphism exactly fits into finding candidates for homonyms, synonyms and similar subgraphs. And the proposed ways to limit the search space for this application of the general problem.

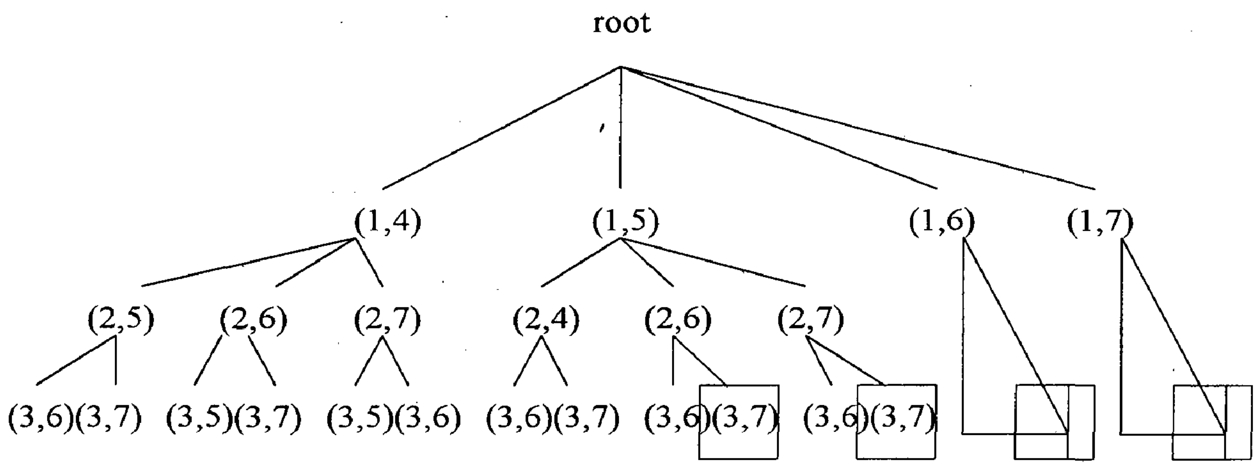


Fig: 4.2 Possible Mappings

4.1.2 Algorithms

Two basic algorithms to find graph isomorphism are Ullmann's Algorithm [6] and A* [7]. They try to efficiently search and prune the search tree, Fig. 4.2

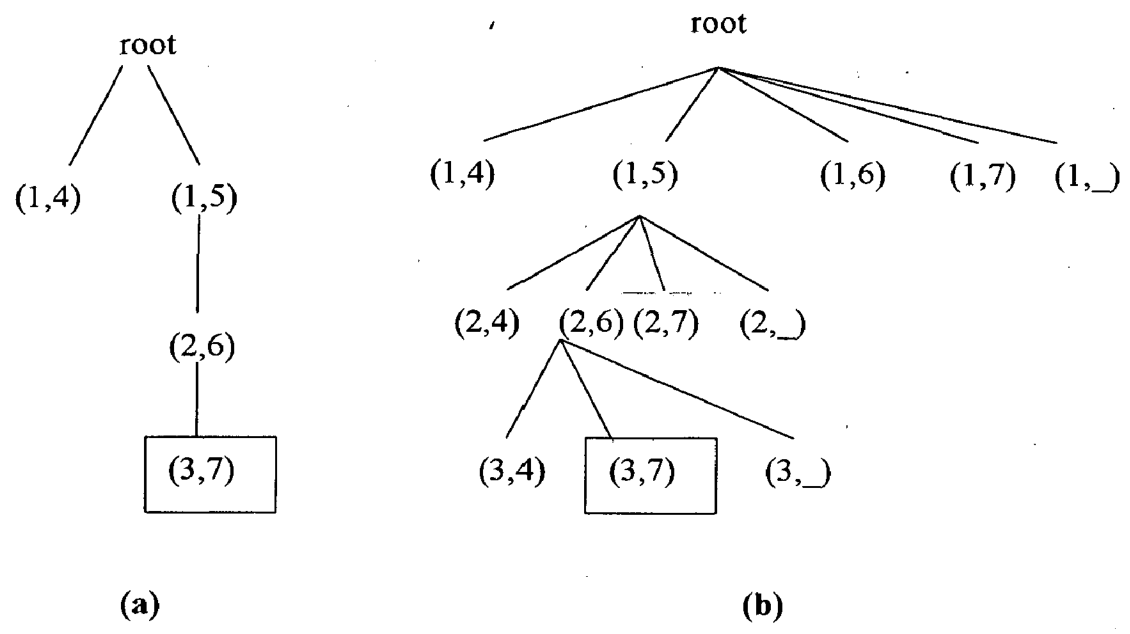


Fig: 4.3 Ullman vs A*

Ullmann's Algorithm searches the tree in depth-first order. As soon as isomorphism condition does not hold in a branch that branch is cut of.

A* searches the tree in breadth-first order. Looking at all branches it decides which node should be expanded. The decision is based on a cost function, which determines a cost for each matching and estimates a cost for matching the unmatched nodes. That is were heuristics about the graphs come into play. A* also provides for inexact matches. Nodes labeled with (X,_) stand for not matching X to anything.

If inexact matches are considered searching in depth-first order is bears the danger of searching branches of infinite length if there is nothing to stop it.

4.1.3 Approach for RDFBrowser

Some of the special properties of RDF graphs can be used in limiting the search space. A nice property of RDF graphs is that only adjacent nodes need to be considered for this task since resources are described by URIs and thus any statement about the same resource will be adjacent in the graph. URIs also provides that nodes with the same URI are mapped onto each other between different graphs. This task is not about finding one perfect match. Finding inexact matches of subgraphs is important because ontologies written by humans will either be known to be the same or won't be the same. It might even be interesting to find isomorphism among metadata, which isn't about the same domain. Then there are few commonly used URIs.

Considering these properties of'the task an algorithm for finding a subgraph isomorphism of specified maximal size m and specified precision p could look as follows:

Input: A query subgraph q and a graph g to which an isomorphism is to be found, m, p.

- Put the nodes in q in list N;
- In N find the nodes whose labels (URIs) also occur in g;
- In N_{URI} find all nodes N_{URI+} with edges that occur in q and g and have the same label;
- Order the nodes N: 1st N_{URI+}, 2nd N_{URI}, then the rest;
- Search the space in depth-first order with the maximum depth m:
    - Try to find matches for the nodes in N (in the obtained order!);
    - If a node n can't be matched match (n,_);
    - If there are more unmatched nodes among the ancestors than precision p allows prune.

If the query graph is specified by giving a node and a depth, to which adjacent nodes are to be considered, then the given node is in the center of that graph.

If the algorithm returns an isomorphism with reasonable precision the node is a candidate for a synonym, if it isn't mapped to the same URI.

If not (even with very low precision), and the node occurs in both files it is a candidate for a homonym.

4.2 Search Using Generic Relevance

The semantic web search is based on certain ontology. Whereas the generic relevance search is on normal web pages and the user performs the relevance filtering. The search engine asks questions until the context and the meaning of the query become clear to the user. Restricts or enlarges the search to more precise or more general subject matters which are more relevant to the user.

a. Analyzing the Meaning of Words

The key terms from the user input are extracted and give the available senses for each keyword. Depending on the user intent of the sense of the keyword the available meanings of the word are obtained. For example the word 'java' may mean coffee or a programming language. The search gives all the available senses for the given word and the user may select his relevant sense for the search.

b. Expanding the Search Operation

If the user wants to include the search with the related concepts also then the search engine enlarges the search inorder to find information related to concepts of a higher level ("hypernyms"). By using this option the search query includes superordinates or superset of the given word.

c. Narrowing the Search Operation

If the user intents to confine the search with in the concept of the key word then the search engine cuts the search to only one branch of the given key word. By using this option only one part of the subset or subordinates of the given word is used for the search.

d. Extracting the Related Paragraphs

The user may not intend to see the entire web page. In such case only the more relevant paragraph for the given query is identified and is displayed to the user. The paragraph operator is provided with similarity list of words. It searches as an AND, but with constraint that the words belong only to some one or n consecutive paragraphs. The paragraph with maximum count for the given list of words is displayed.

Chapter 5

IMPLEMENTATION

5.1 Implementation of Search Based on Semantic Web

The browser used for the semantic web is given the name RDFBrowser. RDFBrowser is a tool for browsing RDF-based metadata including extensions like RDFS, OIL, DAML, but without using their special features. RDFBrowser allows for convenient browsing of the RDF-based metadata and for comparisons between files focusing on helping the user find semantic similarities and differences.

RDFBrowser allows the user to open files containing RDF data and view the triples of the RDF data model. The data is shown in a table containing the triples of the RDF data model. A graphical display of the graph is not chosen in order to support all options described below, because a table offers a better overview over bigger amounts of data and because a table requires much less computation.

By selecting a resource in the table the user can browse through the data. Different triples will be shown in the table according to the selection.

There are 3 basic options for browsing:

a. ***Browsing***: The program browses through the RDF graph. The user can select a resource in the fields “subject” or “object”. If he/she chooses a subject the browser will display all triples in which this resource occurs as an object. If he chooses an object, the browser will display all triples in which this resource occurs as subject. Thus he can move along the edges in the graph. If a user selects a literal, which might occur as object nothing happens.

b. ***Show same***: The user can select a subject, predicate or object and the browser will display all triples in which the selected resource occurs at the same place. This could e.g. be useful to display all subclass relations expressed via the RDF-Schema [RDFS] property rdfs:subClassOf.

c. ***Show all occurrences***: The user can select a subject, predicate or object and the browser will display all triples in which the selection occurs. This is useful to obtain all statements about a certain resource.

The user can open different files and browse each of them with any of those options. There are also 2 global options:

a. ***Single File***: This option is the default. If this is selected the browser shows the behaviour just described.
b. ***Simultaneous***: This option lets the user browse simultaneously through all opened files. He selects an entry in one of the tables and whatever calculation is performed on this data will be performed on all open files. E.g. if subject "a" and the option *browse* is selected in one file the program will look for triples with "a" as subject in all files, which are currently opened.

There is an exception that is considered to be convenient: If there is no such triple in one file the display for that file won't change. It would be more consistent to display an empty table in this case. On the other hand the user would have to reset that table to showing all triples in order to work with this particular one. And there isn't much use in showing an empty table.

Browsing files simultaneously is very useful in order to explore ontologies. It works quite well with ontologies written using DAML, or OIL. A requirement is of course that the same URI are used at least for parts of the ontology. It is easy to find similarities and differences, as long as they are depicted in the URIs used.
In order to take the structure of the RDF graph into account for comparing RDF data a deeper analysis of the structure of the graph has to be performed.

The RDF Browser contains the following components:

a. **Parsing:** The RDF-based data is parsed using the ARP parser, by Jeremy Carrol. This parser is written in Java and depends on the XERCES Java parser. ARP provides means to extract the triples of the RDF data model out of RDF files.
b. **Knowledge Base:** The triples of the RDF data model are assigned into a Prolog knowledge base. RDFBrowser uses AMZI Prolog. It features a Logic Server for Window, Linux, Solaris, and HP/UX Editions. It also provides interfaces to use this server from within Java, C, C++, Delphi and more. Therefore this system fits the need to cooperate with parser and GUI

c. **Graphical User Interface:** It is the one through which user interacts for browsing the files and viewing the triples in the table format. For it Java and its Swing library is used.

5.2 Implementation of Generic Relevance Search

This following figure shows the system architecture of the proposed semantic based search engine.

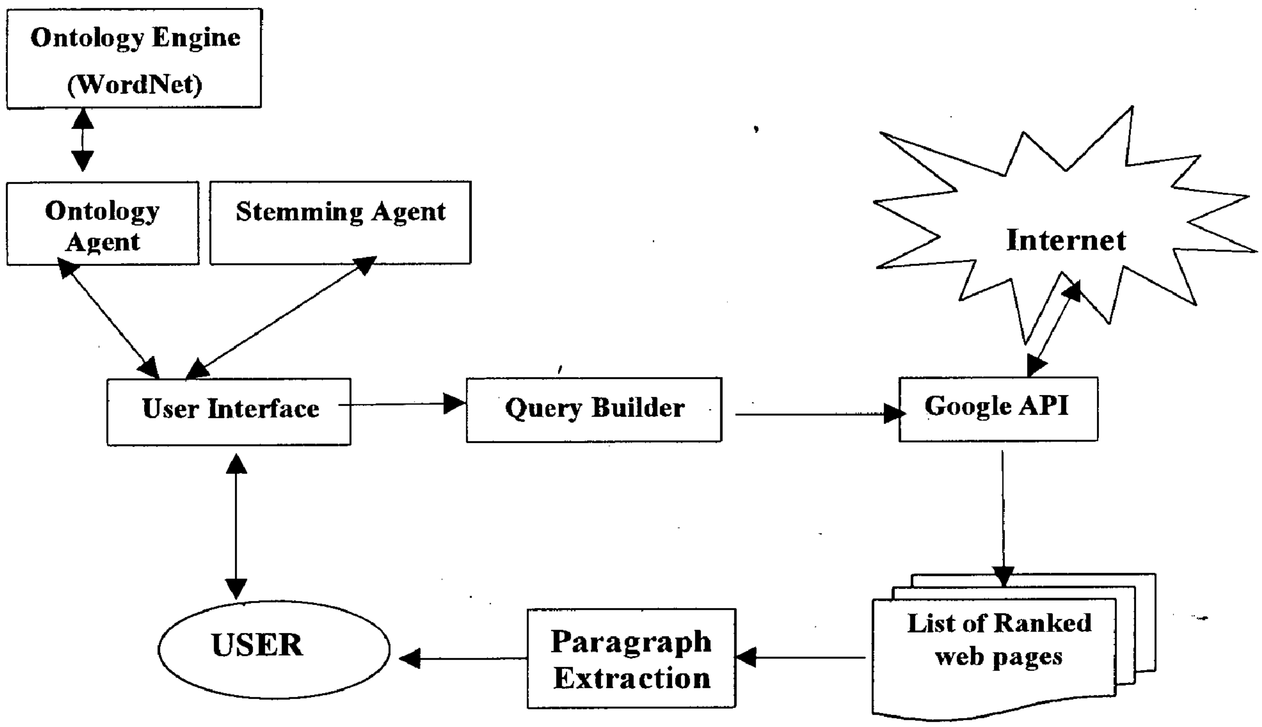


Fig: 5.1 System Architecture

The following are the various components of the proposed search engine.

a. **User Interface**

It is a GUI environment through which user interacts. This includes giving the query, getting the various senses of the query, choosing appropriate sense and displays the final results of the query. User also have the option for expanding or narrow down the search operation. To achieve its goal, it also cooperates with ontology agent, stemming agent and spell checker agent. For it Java and its Swing Library is used.

b. **Stemming Agent**

Stemming agent is developed based on Porter's algorithm [URL19]. The major role of it is to transform the terms in the query to its stemmed terms. The Porter's algorithm is for suffix stripping. For example the words 'connection', 'connected', 'connects' are stripped to the root word 'connect' so that the search ignores the suffixes and includes all forms of the root word. The algorithm is discussed in detail in the Appendix.

c. **Ontology Agent**

Ontology agent is responsible for requesting available concepts of a given term using WordNet and getting the sense of the particular concept. A user interface is created with WordNet using Java. It performs the following steps

- Accessing the WordNet database
- Performing binary search on the index files for the given word
- Obtaining the word category and the synset_offsets
- Accessing the data files and obtaining the word relations.

d. **Query Builder**

The user selects the various options that are available such as for expanding or narrow down the search. Query builder builds the transformed query that is to be submitted for the search by taking the users feedback for the senses available to the query actually given.

e. **Google API**

This accepts the query from the query builder and submits the query to the Google server and provides the related ranked web pages to the user.

f. **Paragraph Extractor**

After obtaining the related web pages if the user wants to view a particular web page instead of displaying the entire web page only the most related paragraph is extracted and displayed to the user. The paragraph operator is provided with similarity list of words. The web page is parsed and a search as an AND operation is performed on each paragraph of the web page. A count of the given words for each paragraph is maintained and the paragraph with maximum count is retrieved and displayed it to the user. The database used is MSAccess.

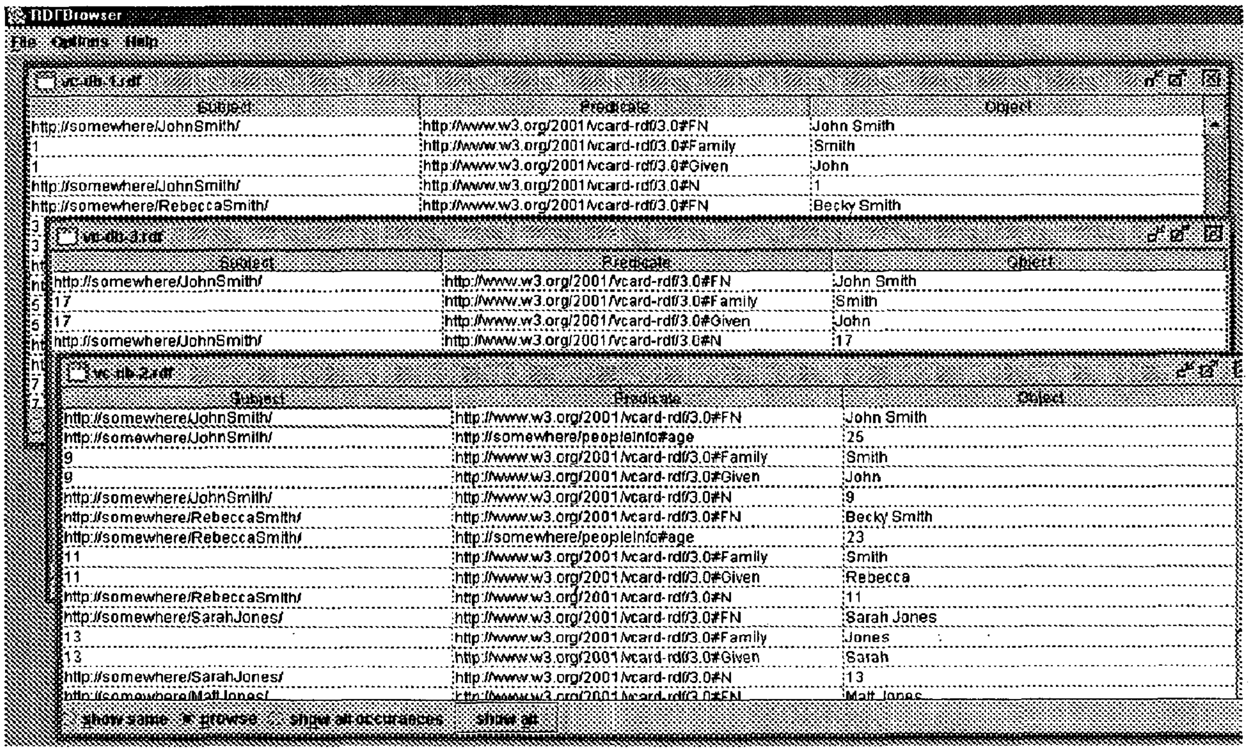
Chapter 6

RESULTS

6.1 Results of RDFBrowser

RDFBrowser is a tool for browsing RDF-based metadata including extensions like RDFS, OIL, DAML, but without using their special features. RDFBrowser allows for convenient browsing of the RDF-based metadata and for comparisons between files focusing on helping the user find semantic similarities and differences.

Fig 6.1 shows the snapshot of RDFBrowser which browses different files simultaneously and displays the RDF content in the triple format (subject, predicate, object).



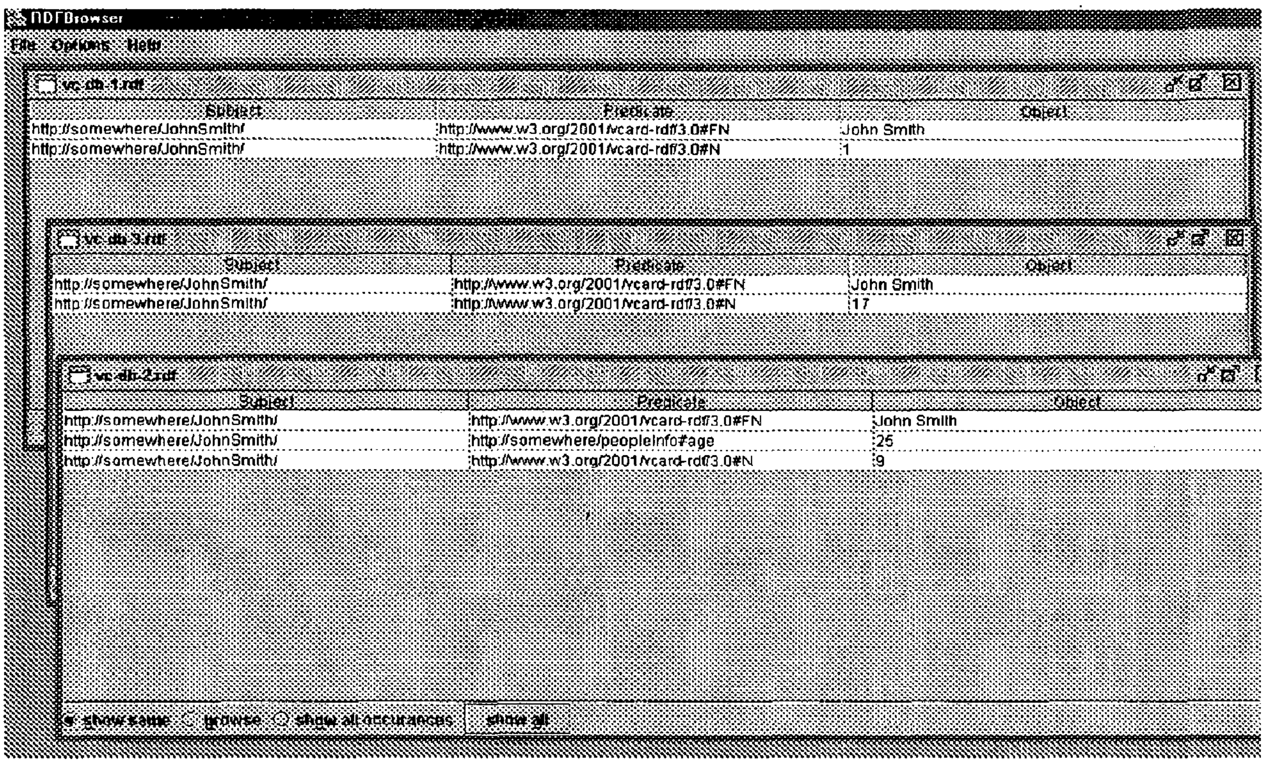
Subject	Predicate	Object
http://somewhere/JohnSmith/	http://www.w3.org/2001/vcard-rdf/3.0#FN	John Smith
1	http://www.w3.org/2001/vcard-rdf/3.0#Family	Smith
1	http://www.w3.org/2001/vcard-rdf/3.0#Given	John
http://somewhere/JohnSmith/	http://www.w3.org/2001/vcard-rdf/3.0#N	1
http://somewhere/RebeccaSmith/	http://www.w3.org/2001/vcard-rdf/3.0#FN	Becky Smith

Subject	Predicate	Object
http://somewhere/JohnSmith/	http://www.w3.org/2001/vcard-rdf/3.0#FN	John Smith
17	http://www.w3.org/2001/vcard-rdf/3.0#Family	Smith
17	http://www.w3.org/2001/vcard-rdf/3.0#Given	John
http://somewhere/JohnSmith/	http://www.w3.org/2001/vcard-rdf/3.0#N	17

Subject	Predicate	Object
http://somewhere/JohnSmith/	http://www.w3.org/2001/vcard-rdf/3.0#FN	John Smith
http://somewhere/JohnSmith/	http://somewhere/peopleInfo#age	25
9	http://www.w3.org/2001/vcard-rdf/3.0#Family	Smith
9	http://www.w3.org/2001/vcard-rdf/3.0#Given	John
http://somewhere/JohnSmith/	http://www.w3.org/2001/vcard-rdf/3.0#N	9
http://somewhere/RebeccaSmith/	http://www.w3.org/2001/vcard-rdf/3.0#FN	Becky Smith
http://somewhere/RebeccaSmith/	http://somewhere/peopleInfo#age	23
11	http://www.w3.org/2001/vcard-rdf/3.0#Family	Smith
11	http://www.w3.org/2001/vcard-rdf/3.0#Given	Rebecca
http://somewhere/RebeccaSmith/	http://www.w3.org/2001/vcard-rdf/3.0#N	11
http://somewhere/SarahJones/	http://www.w3.org/2001/vcard-rdf/3.0#FN	Sarah Jones
13	http://www.w3.org/2001/vcard-rdf/3.0#Family	Jones
13	http://www.w3.org/2001/vcard-rdf/3.0#Given	Sarah
http://somewhere/SarahJones/	http://www.w3.org/2001/vcard-rdf/3.0#N	13
http://somewhere/MattJones/	http://www.w3.org/2001/vcard-rdf/3.0#FN	Matt Jones

Fig: 6.1 Triple Format of RDF content

Fig 6.2 shows the results of the 'show same' option. The user can select subject, predicate or object and the browser will display all triples in which the selected resource occurs at the same place. This is useful to display all subclass relations expressed via the RDF-Schema [RDFS] property `rdfs:subClassOf`. Similarly we can have results for browse and show all occurrences options that are useful to move along the edges of the graph and to obtain all statements abóut a certain resource.



vc-db-1.rdf

Subject	Predicate	Object
http://somewhere/JohnSmith/	http://www.w3.org/2001/vcard-rdf/3.0#FN	John Smith
http://somewhere/JohnSmith/	http://www.w3.org/2001/vcard-rdf/3.0#N	1

vc-db-3.rdf

Subject	Predicate	Object
http://somewhere/JohnSmith/	http://www.w3.org/2001/vcard-rdf/3.0#FN	John Smith
http://somewhere/JohnSmith/	http://www.w3.org/2001/vcard-rdf/3.0#N	17

vc-db-2.rdf

Subject	Predicate	Object
http://somewhere/JohnSmith/	http://www.w3.org/2001/vcard-rdf/3.0#FN	John Smith
http://somewhere/JohnSmith/	http://somewhere/peopleInfo#age	25
http://somewhere/JohnSmith/	http://www.w3.org/2001/vcard-rdf/3.0#N	9

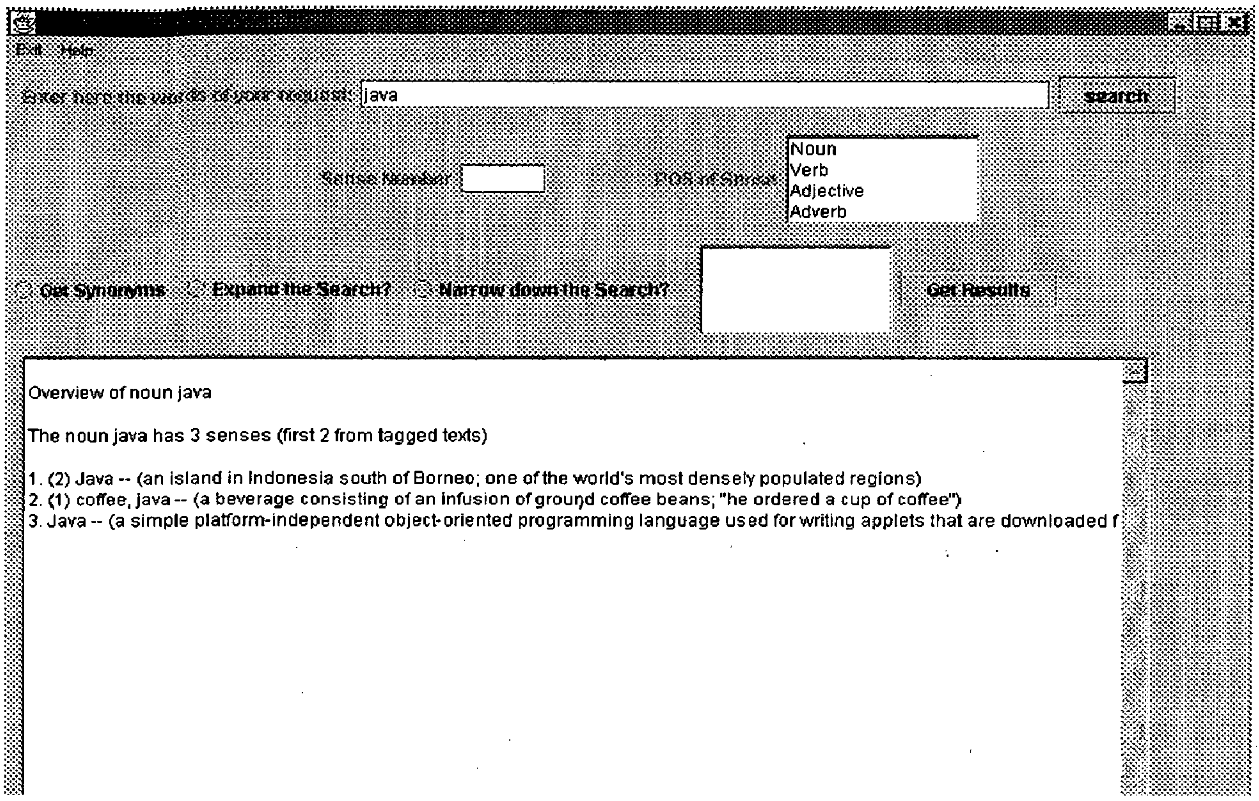
[✓ show same] browse show all occurances

Fig: 6.2 Results for show same option.

6.2 Results of Generic Relevance Search

Personalization is the important feature of this search. User constructs a profile based on his search intention. Using his profile the search is expanded or narrowed down to a particular topic.

Fig 6.3 is the snapshot showing the different senses that are available for a given word. Then the user can select the sense number and the parts of speech of the sense based on his intent of the search. Using this sense one can get the synonyms or expand the search or narrow down the search based on his intention. Fig 6.4 shows the transformed query for expand the search option based on users feedback.

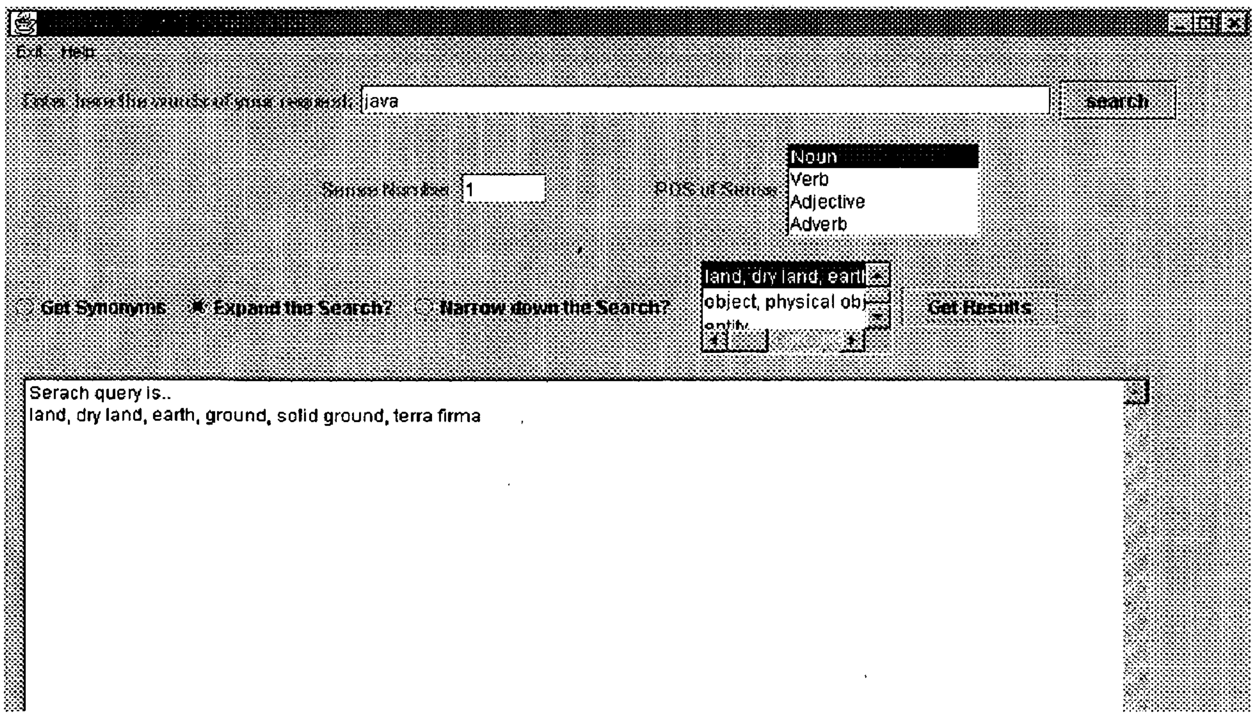


Overview of noun java

The noun java has 3 senses (first 2 from tagged texts)

1. (2) Java -- (an island in Indonesia south of Borneo; one of the world's most densely populated regions)
2. (1) coffee, java -- (a beverage consisting of an infusion of ground coffee beans; "he ordered a cup of coffee")
3. Java -- (a simple platform-independent object-oriented programming language used for writing applets that are downloaded f

Fig: 6.3 Available Senses of the Given Word



Serach query is..
land, dry land, earth, ground, solid ground, terra firma

Fig: 6.4 Transformed Query for Expand the Search Option

Chapter 7

CONCLUSIONS

Semantic Frameworks are arguably an interesting approach for a single organization or a limited topic, especially where particular functionality exists to restrict the ontology and thus allow semiautomatic support for query processing. The approach has also proven useful for navigating subject-specific sites.

A related approach uses machine learning to extract specific information to populate a knowledge base or semantic tags attached to pages. However, neither of these approaches supports the general search with the arbitrary queries that the Web and other large multi-topic document collections require.

The real challenge for Web technologies is scaling the semantic description techniques. In fact, scaling is what the Internet and Internet searches are all about. While a universal ontology would solve the scaling problem, the design of such ontologies is not a simple task, and their integration and maintenance are tedious and error prone. Considering the problems AI researchers have faced in their attempts to build such an ontology—for example, the CYC project—we will likely find ways to automatically generate enough semantic information to yield more meaningful search results long before we define a universal ontology.

Ontologies represent a step toward semi-automated relevance filtering, but they employ a general notion of relevance that applies to the entire user population. Relevance, on the other hand, is typically person-dependent, so personalization will become important in search engines. Most current personalization approaches place the burden of constructing a user profile on the user. This approach is viable in some restricted domains, but it does not scale for Internet search problems. In addition, users may be vocal in demanding what they want, but they often have difficulty defining what they actually need or how they behave.

Using modeling techniques to automatically construct a profile based on user behavior is a scalable approach to personalization. It offers a value-added filtering service that requires no extra work from the user. In fact, users need not even know that an adaptive client-server mechanism exists, though other concerns such as privacy issues might be a reason for revealing it.

Next-Generation Techniques

Large-scale information-retrieval tasks that organize pages hierarchically by topic improve both the relevance of results and the efficient use of computational resources. Returning results by topic enhances the end-user experience and also supports distributed search techniques. In restricted research settings, integrating synonyms and topics into the search system can improve result quality. But such an approach has many pitfalls, and naïve integration can in fact decrease result quality.

One promising way to avoid these problems involves a network of subject-specific nodes in a distributed, hierarchical system (http://cosco.hiit.fi/search). Each "subject node" automatically builds its own hierarchies for terminology, genres, and topics in a *topic map*. Such a map is only a construct, not necessarily a semantic ontology; it exists to power the search and navigation process at a subject-specific site. Eventually, such a "search network" could evolve into a peer-to-peer system with highly distributed query processing using personalized query histories.

Next-Generation Search Example

What could next-generation search facilities do? Consider a general query of a history database, such as "Roman empire." The search engine might suggest specializing the query with a topic such as "military achievements," "politics," or "Christianity," or with a genre such as "discussion groups," "introductions," or "book reviews." At a general level, these specifications reflect suggestions that a knowledgeable librarian might make. A more specific query might identify the topic as free text—for example, "I would like to find out about the daily life of people in the Roman empire"—and select a genre from a pulldown menu—for example, "Research."

The search engine might then order the results under headings such as "entertainment," "moral norms," and "religious ceremonies." Some pages might contain only a subsection relevant to the query. The results list would link directly to this subsection. This presentation reflects analysis of the pages in a collection at a detail level that we could not expect a librarian to master.

Of course, keywords will always remain important, so the search could also include specializations such as "keyword = Bacchus." Suppose a user selects a page that details the religious practices of 1st century B.C.E. priestesses and also considers their portrayal in recent books and films. Then an option to "find more pages like this" could use the page's major semantic terms to index from the full set of 10 million pages in the topic-specific "history engine" into a subset of, say, 400 related pages. The search engine would then rank these 400 using a full-similarity metric and would again break out the results by topic. The approximate full-text similarity matching would then reflect multiple aspects of the selected page's content.

REFERENCES

[1] Vlatko Ceric, *"New Methods and Tools for the World Wide Web Search"*, Journal of Computing and Information Technology – CIT 8 (2000), pp. 267-276

[2] D. Moldovan and R. Mihalcea, "*Using WordNet and Lexical Operators to Improve Internet Searches,*" IEEE Internet Computing, vol. 4(1), 2000, pp. 34-43.

[3] S. Chakrabarti, "*Focused Crawling: a new approach to topic-specific Web resource discovery*", Proceedings of the 8th International World Wide Web Conference (1999) Toronto. (http://www8.org/w8-papers/5a-search-query/crawling/index.html)

[4] R. Allen Wyke, Brad Leupen, Sultan Rehman, *"XML Programming"*, 2002, Microsoft Press, 3rd Edition.

[5] G.A. Miller, "*WORDNET: A Lexical Database for English,*" Comm. ACM, Vol. 2, No.11, Nov. 1995, pp. 39–41.

[6] J. Ullman, "*An Algorithm for subgraph isomorphism",* Journal of the ACM, 23:31-42, 1976.

[7] N. Nilson, "*Principles of Artificial Intelligenc"e.* Tioga, Palo Alto, 1980.

[8] Guarino, N., et al., "*OntoSeek: content-based access to the Web,*" IEEE Intelligent Systems, vol. 14, no. 3,1999, pp. 70-80.

[9] Scime, A. and L. Kerschberg, "*WebSifter: An Ontology-Based Personalizable Search Agent for the Web,*" International Conference on Digital Libraries: Research and Practice, Kyoto Japan, 2000, pp. 493-446.

URLs

[1] W3C Semantic Web, http://www.w3.org/2001/sw/

[2] Tim Berners-Lee, James Hendler, Ora Lassila, "*The Semantic Web*", Scientific American, http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21

[3] Sean B. Palmer, "*The Semantic Web: An Introduction*", 2001, http://infomesh.net/2001/swintro/

[4] Eric Prudhommeaux, "*Presentation of W3C and Semantic Web*", 2001, http://www.w3.org/2001/Talks/0710-ep-grid

[5] Tim Berners-Lee, "*W3C Naming and Addressing Overview (URIs. URLs, ...)*", http://www.w3.org/Addressing/

[6] W3C, "*W3C Resource Description Framework (RDF)*", http://www.w3.org/RDF/

[7] The DARPA Agent Markup Language (DAML), http://www.daml.org/

[8] Aaron Swartz, "*The Semantic Web (for Web Developers)*", May 2001, http://logicerror.com/semanticWeb-webdev

[9] Sandro Hawke, "*How the Semantic Web Works*", April 2002, http://www.w3.org/2002/03/semweb/

[10] IETF, Hypertext Transfer Protocol -- HTTP/1.1 - Draft Standard RFC 2616, http://www.ietf.org/rfc/rfc2616.txt

[11] W3Schools (http://www.w3schools.com), "*XML Schema Tutorial*", (2001) http://www.w3schools.com/schema/schema_intro.asp

[12] Tim Bray, "*What is RDF?*", http://www.xml.com/lpt/a/2001/01/24/rdf.html

[13] W3C, RDF Primer, W3C Working Draft 23 January 2003, http://www.w3.org/TR/2003/WD-rdf-primer-20030123/

[14] W3C, Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222

[15] W3C, RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft 23 January 2003, http://www.w3.org/TR/rdf-schema/

[16] Roxane Ouellet, Uche Ogbuji, Introduction to DAML: Part I (2002), http://www.xml.com/lpt/a/2002/01/30/daml1.html

[17] Roxane Ouellet, Uche Ogbuji, Introduction to DAML: Part II (2002), http://www.xml.com/lpt/a/2002/03/13/daml.html

[18] Adam Pease, Why Use DAML? (10 April, 2002), http://www.daml.org/2002/04/why.html

[19] Porter, M., "*An Algorithm for Suffix Stripping*", http://www.muscat.co.uk/~martin/def.txt.

APPENDIX

Porter's Algorithm for Suffix Stripping

Removing suffixes by automatic means is an operation, which is especially useful in the field of information retrieval. In a typical IR environment, one has a collection of documents, each described by the words in the document title and possibly by words in the document abstract. Ignoring the issue of precisely where the words originate, we can say that a vector of words, or *terms* represents a document. Terms with a common stem will usually have similar meanings, for example:

CONNECT

CONNECTED

CONNECTING

CONNECTION

CONNECTIONS

Frequently, the performance of an IR system will be improved if term groups such as this are conflated into a single term. This may be done by removal of the various suffixes -ED, -ING, -ION, IONS to leave the single term CONNECT. In addition, the suffix stripping process will reduce the total number of terms in the IR system, and hence reduce the size and complexity of the data in the system, which is always advantageous.

Many strategies for suffix stripping have been reported in the literature. The nature of the task will vary considerably depending on whether a stem dictionary is being used, whether a suffix list is being used, and of course on the purpose for which the suffix stripping is being done. Assuming that one is not making use of a stem dictionary, and that the purpose of the task is to improve IR performance, the suffix stripping program will usually be given an explicit list of suffixes, and, with each suffix, the criterion under which it may be removed from a word to leave a valid stem. This is the approach adopted here. The main merits of the present program are that it is small (less than 400 lines of BCPL), fast (it will process a vocabulary of 10,000 different words in about 8.1 seconds on the IBM 370/165 at Cambridge University), and reasonably simple. At any rate, it is simple enough to be described in full as an algorithm in this paper. (The present version in BCPL is freely available from the author. BCPL is itself available on a wide range of different computers, but anyone wishing to use the program should have little difficulty in coding it up in other programming languages.) Given the speed of the program, it would be quite realistic to apply it to every word in a large file of continuous text, although for historical reasons we have found it convenient to apply it only to relatively small vocabulary lists derived from continuous text files.

In any suffix stripping program for IR work, two points must be borne in mind. Firstly, the suffixes are being removed simply to improve IR performance, and not as a linguistic exercise. This means that it would not be at all obvious under what circumstances a suffix should be removed, even if we could exactly determine the suffixes of a word by automatic means.

Perhaps the best criterion for removing suffixes from two words W1 and W2 to produce a single stem S, is to say that we do so if there appears to be no difference between the two statements `a document is about W1' and `a document is about W2'. So if W1=`CONNECTION' and W2=`CONNECTIONS' it seems very reasonable to conflate them to a single stem. But if W1=`RELATE' and W2=`RELATIVITY' it seems perhaps unreasonable, especially if the document collection is concerned with theoretical physics. (It should perhaps be added that RELATE and RELATIVITY *are* conflated together in the algorithm described here.) Between these two extremes there is a continuum of different cases, and given two terms W1 and W2, there will be some variation in opinion as to whether they should be conflated, just as there is with deciding the relevance of some document to a query. The evaluation of the worth of a suffix stripping system is correspondingly difficult.

The second point is that with the approach adopted here, i.e. the use of a suffix list with various rules, the success rate for the suffix stripping will be significantly less than 100% irrespective of how the process is evaluated. For example, if SAND and SANDER get conflated, so most probably will WAND and WANDER. The error here is that the -ER of WANDER has been treated as a suffix when in fact it is part of the stem. Equally, a suffix may completely alter the meaning of a word, in which case its removal is unhelpful. PROBE and PROBATE for example, have quite distinct meanings in modern English. (In fact these would not be conflated in our present algorithm.) There comes a stage in the development of a suffix stripping program where the addition of more rules to increase the performance in one area of the vocabulary causes an equal degradation of performance elsewhere. Unless this phenomenon is noticed in time, it is very easy for the program to become much more complex than is really necessary. It is also easy to give undue emphasis to cases, which appear to be important, but which turn out to be rather rare. For example, cases in which the root of a word changes with the addition of a suffix, as in DECEIVE/DECEPTION, RESUME/RESUMPTION, INDEX/INDICES occur much more rarely in real vocabularies than one might at first suppose. In view of the error rate that must in any case be expected, it did not seem worthwhile to try and cope with these cases.

It is not obvious that the simplicity of the present program is any demerit. In a test on the well-known Cranfield 200 collection it gave an improvement in retrieval performance when compared with a very much more elaborate program which has been in use in IR research in Cambridge since 1971. The test was done as follows: the words of the titles and abstracts in the documents were passed through the earlier suffix stripping system, and the result is stems were used to index the documents. The words of the queries were reduced to stems in the same way, and the documents were ranked for each query using term coordination matching of query against document. From these rankings, recall and precision values were obtained using the standard recall cutoff method. The entire process was then repeated using the suffix stripping system described in this paper, and the results were as follows:

earlier system		present system	
precision	recall	precision	recall
0	57.24	0	58.60
10	56.85	10	58.13
20	52.85	20	53.92

earlier system		present system	
precision	recall	precision	recall
30	42.61	30	43.51
40	42.20	40	39.39
50	39.06	50	38.85
60	32.86	60	33.18
70	31.64	70	31.19
80	27.15	80	27.52
90	24.59	90	25.85
100	24.59	100	25.85

Cleary, the performance is not very different. The important point is that the earlier, more elaborate system certainly performs no better than the present, simple system.

(This test was done by prof. C.J. van Rijsbergen.)

The Algorithm:

To present the suffix stripping algorithm in its entirety we will need a few definitions.

A *consonant* in a word is a letter other than A, E, I, O or U, and other than Y preceded by a consonant. (The fact that the term `consonant' is defined to some extent in terms of itself does not make it ambiguous.) So in TOY the consonants are T and Y, and in SYZYGY they are S, Z and G. If a letter is not a consonant it is a *vowel.*

A consonant will be denoted by c, a vowel by v. A list ccc... of length greater than 0 will be denoted by C, and a list vvv... of length greater than 0 will be denoted by V. Any word, or part of a word, therefore has one of the four forms:

CVCV ... C

CVCV ... V

VCVC ... C

VCVC ... V

These may all be represented by the single form

[C]VCVC ... [V]

where the square brackets denote arbitrary presence of their contents.

Using (VC){m} to denote VC repeated m times, this may again be written as

[C](VC){m}[V].

m will be called the \measure\ of any word or word part when represented in this form. The case m = 0 covers the null word. Here are some examples:

m=0	TR, EE, TREE, Y, BY.
m=1	TROUBLE, OATS, TREES, IVY.
m=2	TROUBLES, PRIVATE, OATEN, ORRERY.

The \rules\ for removing a suffix will be given in the form

(condition) S1 -> S2

This means that if a word ends with the suffix S1, and the stem before S1 satisfies the given condition, S1 is replaced by S2. The condition is usually given in terms of m, e.g.

(m > 1) EMENT ->

Here S1 is `EMENT' and S2 is null. This would map REPLACEMENT to REPLAC, since REPLAC is a word part for which m = 2.

The `condition' part may also contain the following:

*S - the stem ends with S (and similarly for the other letters).

v - the stem contains a vowel.

*d - the stem ends with a double consonant (e.g. -TT, -SS).

*o - the stem ends cvc, where the second c is not W, X or Y (e.g.-WIL, -HOP).

And the condition part may also contain expressions with *and*, *or* and *not*, so that.

(m>1 and (*S or *T))

tests for a stem with m>1 ending in S or T, while

(*d and not (*L or *S or *Z)) ,

tests for a stem ending with a double consonant other than L, S or Z. Elaborate conditions like this are required only rarely.

In a set of rules written beneath each other, only one is obeyed, and this will be the one with the longest matching S1 for the given word. For example, with

SSES -> SS

IES -> I

SS -> SS

S ->

(here the conditions are all null) CARESSES maps to CARESS since SSES is the longest match for S1. Equally CARESS maps to CARESS (S1=`SS') and CARES to CARE (S1=`S').

In the rules below, examples of their application, successful or otherwise, are given on the right in lower case. The algorithm now follows:

Step 1a

SSES -> SS	caresses -> caress
IES -> I	ponies -> poni
	ties -> ti
SS -> SS	caress -> caress
S ->	cats -> cat

Step 1b

(m>0) EED -> EE	feed -> feed
	agreed -> agree
(*v*) ED ->	plastered -> plaster
	bled -> bled
(*v*) ING ->	motoring -> motor
	sing -> sing

If the second or third of the rules in Step 1b is successful, the following is done:

AT -> ATE	conflat(ed) -> conflate
BL -> BLE	troubl(ed) -> trouble
IZ -> IZE	siz(ed) -> size
(*d and not (*L or *S or *Z)) -> single letter	
	hopp(ing) -> hop
	tann(ed) -> tan
	fall(ing) -> fall
	hiss(ing) -> hiss
	fizz(ed) -> fizz
(m=1 and *o) -> E	fail(ing) -> fail
	fil(ing) -> file

The rule to map to a single letter causes the removal of one of the double letter pair. The -E is put back on -AT, -BL and -IZ, so that the suffixes -ATE, -BLE and -IZE can be recognized later. This E may be removed in step 4.

Step 1c

(*v*) Y -> I	happy -> happi
	sky -> sky

Step 1 deal with plurals and past participles. The subsequent steps are much more straightforward.

Step 2

(m>0) ATIONAL -> ATE	relational -> relate
(m>0) TIONAL -> TION	conditional -> condition
	rational -> rational
(m>0) ENCI -> ENCE	valenci -> valence
(m>0) ANCI -> ANCE	hesitanci -> hesitance
(m>0) IZER -> IZE	digitizer -> digitize
(m>0) ABLI -> ABLE	conformabli -> conformable
(m>0) ALLI -> AL	radicalli -> radical
(m>0) ENTLI -> ENT	differentli -> different
(m>0) ELI -> E	vileli -> vile
(m>0) OUSLI -> OUS	analogousli -> analogous
(m>0) IZATION -> IZE	vietnamization -> vietnamize
(m>0) ATION -> ATE	predication -> predicate
(m>0) ATOR -> ATE	operator -> operate
(m>0) ALISM -> AL	feudalism -> feudal
(m>0) IVENESS -> IVE	decisiveness -> decisive
(m>0) FULNESS -> FUL	hopefulness -> hopeful
(m>0) OUSNESS -> OUS	callousness -> callous

(m>0) ALITI -> AL	formaliti -> formal
(m>0) IVITI -> IVE	sensitiviti -> sensitive
(m>0) BILITI -> BLE	sensibiliti -> sensible

The test for the string S1 can be made fast by doing a program switch on the penultimate letter of the word being tested. This gives a fairly even breakdown of the possible values of the string S1. It will be seen in fact that the S1-strings in step 2 are presented here in the alphabetical order of their penultimate letter. Similar techniques may be applied in the other steps.

Step 3

(m>0) ICATE -> IC	triplicate -> triplic
(m>0) ATIVE ->	formative -> form
(m>0) ALIZE -> AL	formalize -> formal
(m>0) ICITI -> IC	electriciti -> electric
(m>0) ICAL -> IC	electrical -> electric
(m>0) FUL ->	hopeful -> hope
(m>0) NESS ->	goodness -> good

Step 4

(m>1) AL ->	revival -> reviv
(m>1) ANCE ->	allowance -> allow
(m>1) ENCE ->	inference -> infer
(m>1) ER ->	airliner -> airlin
(m>1) IC ->	gyroscopic -> gyroscop
(m>1) ABLE ->	adjustable -> adjust
(m>1) IBLE ->	defensible -> defens
(m>1) ANT ->	irritant -> irrit

(m>1) EMENT ->	replacement -> replac
(m>1) MENT ->	adjustment -> adjust
(m>1) ENT ->	dependent -> depend
(m>1 and (*S or *T)) ION ->	adoption -> adopt
(m>1) OU ->	homologou -> homolog
(m>1) ISM ->	communism -> commun
(m>1) ATE ->	activate -> activ
(m>1) ITI ->	angulariti -> angular
(m>1) OUS ->	homologous -> homolog
(m>1) IVE ->	effective -> effect
(m>1) IZE ->	bowdlerize -> bowdler

The suffixes are now removed. All that remains is a little tidying up.

Step 5a

(m>1) E ->	probate -> probat
	rate -> rate
(m=1 and not *o) E ->	cease -> ceas

Step 5b

(m > 1 and *d and *L) -> single letter

	controll -> control
	roll -> roll

The algorithm is careful not to remove a suffix when the stem is too short, the length of the stem being given by its measure, m. There is no linguistic basis for this approach. It was merely observed that m could be used quite effectively to help decide whether or not it was wise to take off a suffix. For example, in the following two lists:

list A	list B
RELATE	DERIVATE
PROBATE	ACTIVATE
CONFLATE	DEMONSTRATE
PIRATE	NECESSITATE
PRELATE	RENOVATE

-ATE is removed from the list B words, but not from the list A words. This means that the pairs DERIVATE/DERIVE, ACTIVATE/ACTIVE, DEMONSTRATE/DEMONSTRABLE, NECESSITATE/NECESSITOUS, will conflate together. The fact that no attempt is made to identify prefixes can make the results look rather inconsistent. Thus PRELATE does not lose the -ATE, but ARCHPRELATE becomes ARCHPREL. In practice this does not matter too much, because the presence of the prefix decreases the probability of an erroneous conflation.

Complex suffixes are removed bit by bit in the different steps. Thus GENERALIZATIONS is stripped to GENERALIZATION (Step 1), then to GENERALIZE (Step 2), then to GENERAL (Step 3), and then to GENER (Step 4). OSCILLATORS is stripped to OSCILLATOR (Step 1), then to OSCILLATE (Step 2), then to OSCILL (Step 4), and then to OSCIL (Step 5). In a vocabulary of 10,000 words, the reduction in size of the stem was distributed among the steps as follows:

Suffix stripping of a vocabulary of 10,000 words

Number of words reduced in step 1:	3597
" 2:	766
" 3:	327
" 4:	2424
" 5:	1373
Number of words not reduced:	3650

The resulting vocabulary of stems contained 6370 distinct entries. Thus the suffix stripping process reduced the size of the vocabulary by about one third.

(This algorithm was originally published by M.F.Porter, in *Program, 14* no. 3, pp 30-137, July 1980. Later a few typos have been corrected.)