SEMANTIC BROWSER

by

BILAL GONEN

(Under the Direction of Amit Sheth)

Relationships are keys to the semantics and hence the Semantic Web. One interesting way to exploit relationships is to link documents such that terms in the documents are semantically related through well defined relationships. In his 1945 article, Dr. Vannevar Bush posited the idea of creating a device capable of recording all of human knowledge. Dr. Bush suggested that such a device would allow its user to traverse a space of documents by following a “trail of associations” in the user’s mind. The assumption however is that links of the Web are meaningful to the user. This assumption certainly does not hold for the eight billion pages on the Web. We consider a scenario where the user has some prior knowledge of the domain. Our approach to building such a system relies on two aspects of Semantic Web: (a) semantic metadata for documents, and (b) populated ontology with relationship instances.

INDEX WORDS: Semantic web, Semantic Browser, UMLS, PubMed, MeSH, AJAX.

SEMANTIC BROWSER

by

BILAL GONEN

B.E., Sakarya University, Turkey, 2002

A Thesis Submitted to The Graduate Faculty of The University of Georgia in Partial Fulfillment

of Requirements for The Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2006

SEMANTIC BROWSER

by

BILAL GONEN

ACKNOWLEDGMENTS

I would like to acknowledge the help of my major Professor Amit Sheth. He was the major figure for my education in UGA. I thank Profs. Miller and Arpinar for their review and guidance related to this thesis. My special thanks to Cartic Ramakrishnan who has been my senior collaborator in this endeavor. He provided the domain knowledge, ontology and the data set, which made it possible for me to demonstrate my tool. I also would like to thank to my colleagues in the LSDIS Lab, especially Matthew Evanson and Aleman Boanerges who helped me during my thesis in various ways.

TABLE OF CONTENTS

Page

1. INTRODUCTION 1

2. BUILDING RELATIONSHIP WEB 6

3. SYSTEM ARCHITECTURE 10

3.1. ONTOLOGY USED IN THE PROCESS 10

3.2 DATASET USED IN THE PROJECT 13

3.3 INDEXING THE ARTICLES IN THE DATASET 16

3.4 USER INTERFACE 17

3.5 SERVER-SIDE COMPONENTS 17

4. SEMANTIC BROWSER APPLICATION 19

5. RELATED WORK 25

6. CONCLUSIONS AND FUTURE WORK 28

7. REFERENCES 30

APPENDICES 33

1. INTRODUCTION

There have been significant research activities in document categorization. A number of tools have been built that provide users to browse among the classified collection of documents. In order to give the user ability to pick the most relevant documents which are classified under a category, several methods have being used to rank them based on their representative values, i.e., term frequency, inverse document frequency, etc. In most of these tools, the classification of the documents is built as a taxonomy. There is no connection between the different nodes in the taxonomy other than hierarchy-relationships. Consequently such a document organization restricts the users to browse documents under a particular category. To get to a document on a related topic the user has to navigate up the hierarchy and back down to the relevant topic.

For instance, consider a user reading an article about the disease "measles". The user would like to read articles which talk about some drugs that help in curing measles. For instance, there is a "cure" relationship between "egg plant seeds" and "measles". Here is a quotation from the home-remedies web site: "The seeds of the egg plant are a stimulant. Intake of half to one gram of these seeds daily for three days will help develop immunity against measles for one year."[3] Although these two concepts "egg plant seeds" and "measles" are related to each other by an important relationship between them, how likely is it that they may be classified under the same category? Availability of such “related” information will depend on whether the author of the text included such information. Without having the information that a relationship “cures” links the concepts, or having a means to use this information, a document categorization tool would not put documents about these two concepts under the same category, because one of them is a disease, and the other one in a plant. A categorization system arguably should not put such “related” documents in the same category, but the ability to navigate from a document about “measles” to one about a cure is a very useful one to have. The egg plant seeds are not the only things helping in remedy of measles. The same webpage [3] mentions that "application of mud packs" also helps in remedy for measles. Therefore, would not it be nice to suggest the user to read about articles about "egg plant seeds", "mud packs", "barley", "turmeric", and perhaps some others, by also giving the "cures" relationship, thus telling the reason why some articles about these topics are offered? Also, before offering the articles of topics which cures the measles, would not it be better to offer the user the relationships about the measles, first? Perhaps, the user may not be interested in reading topics about curing the measles, but may be interested in reading topics about causes of measles. By choosing this relation, totally different topics would be offered to the user, instead of egg plant seeds, barley, etc. which help cure the measles.

The above discussion demonstrates the need for using some kind of knowledge base which includes the entities along with relationships between them is very obvious. Here is where the Semantic Web comes in. The Semantic Web has gained much interest during the past few years. Data typically published on the web is human understandable and is meant for human consumption. However, the Semantic Web makes such data machine-understandable. Associating formal semantics with data, and using it in making search, browsing and analysis more intelligent and precise, can also lead to saving time for a user who would otherwise have to peruse more data to get all the information he or she seeks.

When we see a word in the text, for instance "America", human cognitive processes associate meaning with the term. However, a machine does not know that “America” is a country, located in North America, has a border with Canada, etc. As humans, we know that America is (likely or certainly) the USA, which is a country. We need to attach the property or the relationship "country" to the word "America" somehow. Also we know that America, USA, United States, US, etc. all refer to the same thing. Should we also attach such information to the word "America"? How about the concepts that have some relationships with America? Georgia in the relationship of being a state, or George W. Bush, in the relationship of being the President, for example. It would be useful for us to attach this information to the word America in text somehow to have the machine find the relevant information for the user. Instead of attaching all the information we know about a word in the plain text, some external knowledge bases (parts of what is called ontologies) are used in the Semantic Web. In the ontology, we have class-subclass hierarchy, such as apple is subclass of fruit, and fruit is subclass of food, and so on. Taxonomy also has these hierarchical relations. The difference between ontologies and taxonomies is that ontologies have named relationships between the classes, even if they do not have hierarchical relationship. However, taxonomy does not have such relationships besides hierarchical relationships.

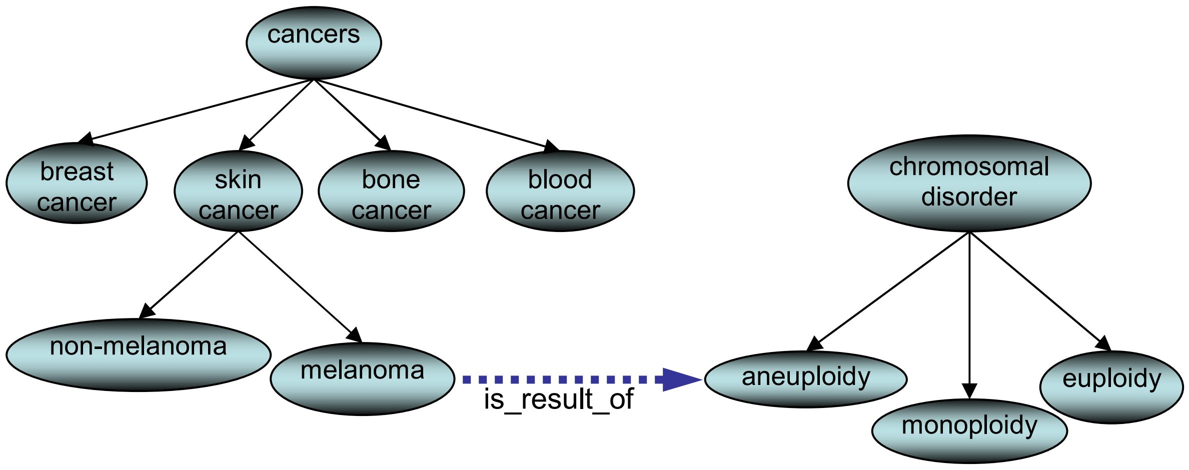


Figure 1 Relationship in the ontology

In his 1945 article, Dr. Vannevar Bush, then the director of the Office of Scientific Research and Development, pointed out the importance of relationships between two different concepts even if they are not connected with a class-subclass relationship in the taxonomy; "When data of any sort are placed in storage, they are filed alphabetically or numerically, and information is found (when it is) by tracing it down from subclass to subclass. It can be in only one place, unless duplicates are used; one has to have rules as to which path will locate it, and the rules are cumbersome. Having found one item, moreover, one has to emerge from the system and re-enter on a new path.

The human mind does not work that way. It operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain." [4]

Our Semantic Browser tool enables easier navigation with relationships as opposed to hyperlinks. In their paper, Logical Information Modeling of Web-accessible Heterogeneous Digital Assets [5], Amit Sheth, and Kshitij Shah defined hyperlinks as "physical (hard) relationships" and semantic relationships in the Relationship Web as "virtual links".

In this thesis, we also show that more relevant documents are returned by using the virtual links as opposed to hyperlinks which do not have semantic relationship between the documents. Processing the documents in our dataset by using our ontology, we built a relationship web in which documents are connected to each other with relationships. The advantage of these virtual connections is that a user can navigate from one document to another using relationships, even if there is not any hyperlink between those documents.

Our work in this thesis is one of the earliest attempts at utilizing the semantics of the relationships to support browsing and navigation of a document space.

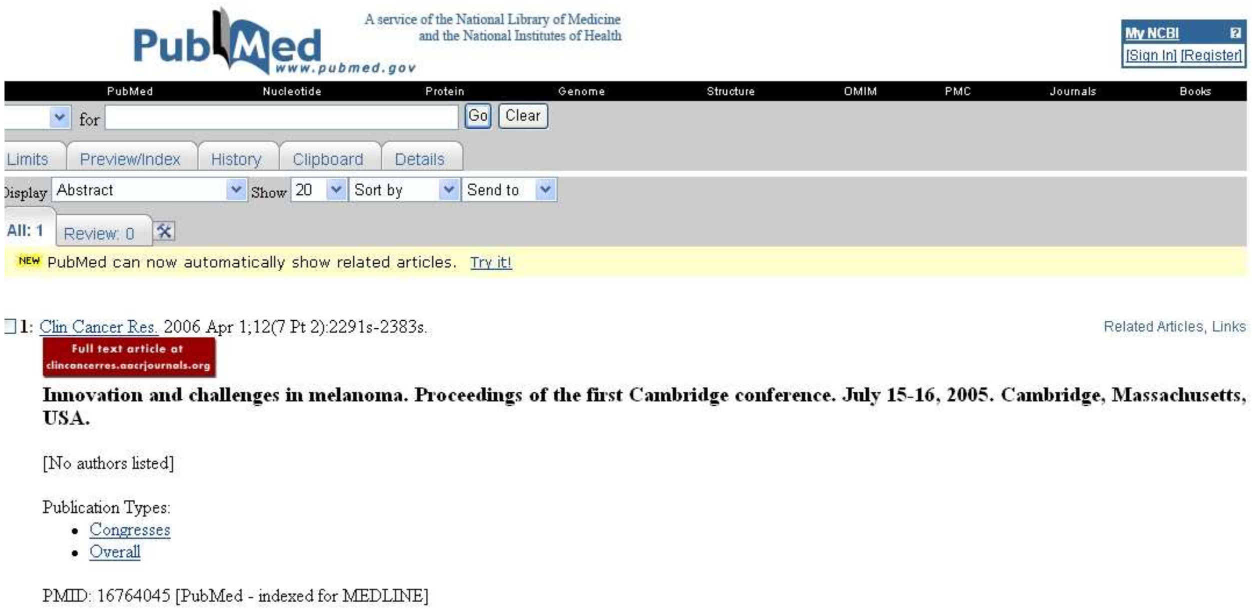
In this thesis, we describe the concept “Relationship Web (Section 2). Then, we explain the system architecture in Section 3. In that section, we describe the ontology used in the project, dataset, user interface and server-side components. Section 4 shows the “Semantic Browser” tool [31]. In section 5, we review related work. In section 6, conclusions and future work are explained.

2. BUILDING RELATIONSHIP WEB[1]

Given a set of documents along with an ontology schema and a set of entity instances; our goal is to create a web of documents linked to each other by named relationships. Any two documents may not be connected to each other by physical links (HREF), but may contain terms which are related to each other based on the ontology used. Without building the Relationship Web, the user does not have a way to navigate semantically connected documents unless there is at least one physical link between those documents. Even with the existence of such a physical link, the relationship between the hyperlinked term and the target page is based on the interpretation of the user. Although the relationship is human understandable, it is not machine understandable.

Some web sites have some system to return related articles to the user. However, the relationships between those related articles are not named. Another issue is that the user may not be interested in reading the articles related to the article she reads at that moment, but may be interested in reading articles related to any particular term in the article she reads.

[1] Relationship Web is a term introduced by Prof. Amit Sheth. This thesis deals with only a narrow aspect of the broader theme.


PubMed
www.pubmed.gov
A service of the National Library of Medicine and the National Institutes of Health
My NCBI
[Sign In] [Register]
PubMed Nucleotide Protein Genome Structure OMIM PMC Journals Books
for
Go Clear
Limits Preview/Index History Clipboard Details
Display Abstract Show 20 Sort by Send to
All: 1 Review: 0
Full text article at clincancerres.aacrjournals.org
Related Articles, Links

NEW PubMed can now automatically show related articles. Try it!

1: Clin Cancer Res. 2006 Apr 1;12(7 Pt 2):2291s-2383s.

Innovation and challenges in melanoma. Proceedings of the first Cambridge conference. July 15-16, 2005. Cambridge, Massachusetts, USA.

[No authors listed]

Publication Types:
- Congresses
- Overall

PMID: 16764045 [PubMed - indexed for MEDLINE]

Figure 2: A screenshot from PubMed Web site

For instance let us consider a scenario that a user reads a medical article in the PubMed [6] Web site. She reads a sentence that contains "melanoma". Melanoma is a type of skin cancer. In the page where she reads the article abstract, the web site includes a hyperlink saying "Related Articles". The user may not be interested in the articles related to the main concept of the article she reads at that time, but may be interested in the article related to the particular term "melanoma" that she has just encountered in the article abstract. If the term "melanoma" in the sentence is not hyperlinked, then the user has to type the term "melanoma" in the search box at the top of the page, to read articles about the melanoma. What if she is not interested in articles about the melanoma, but is interested learning about the causes of the "melanoma"? If the user does not know the causes of the melanoma already, there is no way that she can reach to the articles which talk about those causes in the current system. She needs to know the causes beforehand, so that she can type their names in the search box.

Imagine that in order to return related terms to user, a Web site uses a statistical mechanism. Such statistical mechanisms may bind two terms together as “related” just because either they occur in the same sentences frequently. If we consider the recommendation systems on the online stores; they relate some products together, just because they are frequently purchased in the same transactions. According to such statistical systems, the only relationship between those products is statistical proximity. Instead of giving the user related terms only, it is necessary for the user to know the relationships also, because the user may not be interested in spending her time to read articles about “aneuploidy”, unless she knows that aneuploidy is a result of melanoma. By using our approach, however, for the chosen term “melanoma”, the user is offered several relationships, such as “affects”, “co occurs with”, “occurs in”, and “is result of”. By choosing the relationship “is result of”, the user can get the terms which are the result of “melanoma”.

As can be seen in the scenario above, no physical link (hyperlink) is needed to navigate to “semantically” related documents. Also, no prior knowledge is needed by the user to know what the results of “melanoma” are.

The critical research issue, however, is to identify which of the prohibitively large number of relationships are more relevant than others. We demonstrate this capability by developing an application that allows users to browse documents by following chains of named relationships much the same way we follow hyperlink today. As a starting point, we test a subset of PubMed [6] abstracts linked to each other by named relationships. The named entities at the instance level are associated with some schema class types at the schema level of the ontology. The instances are usually associated with one, two, or three class types at the schema level. Because we do not yet have relationships between concepts at the instance level, we look at the relationships between the types of those instances. Then we use these relationships to build our relationship web.

3. SYSTEM ARCHITECTURE

The system architecture consists of several subcomponents. In this chapter, we will explain the ontology used in the project, dataset, user interface and server-side components.

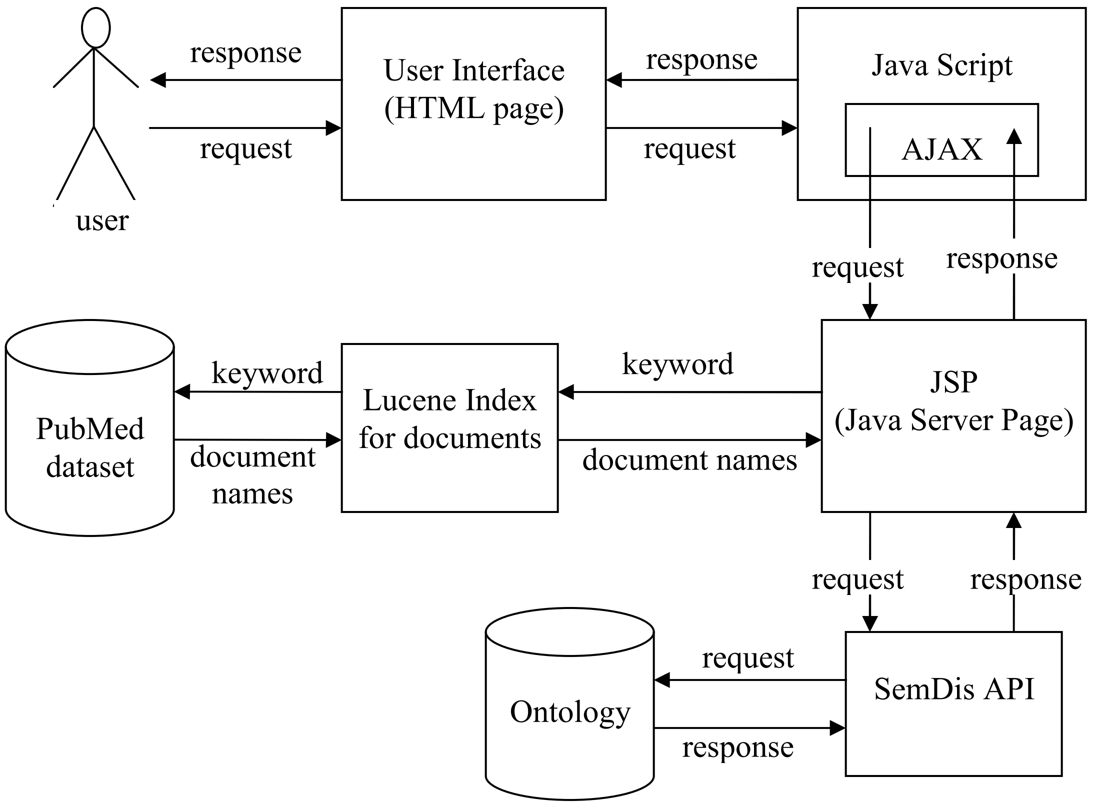
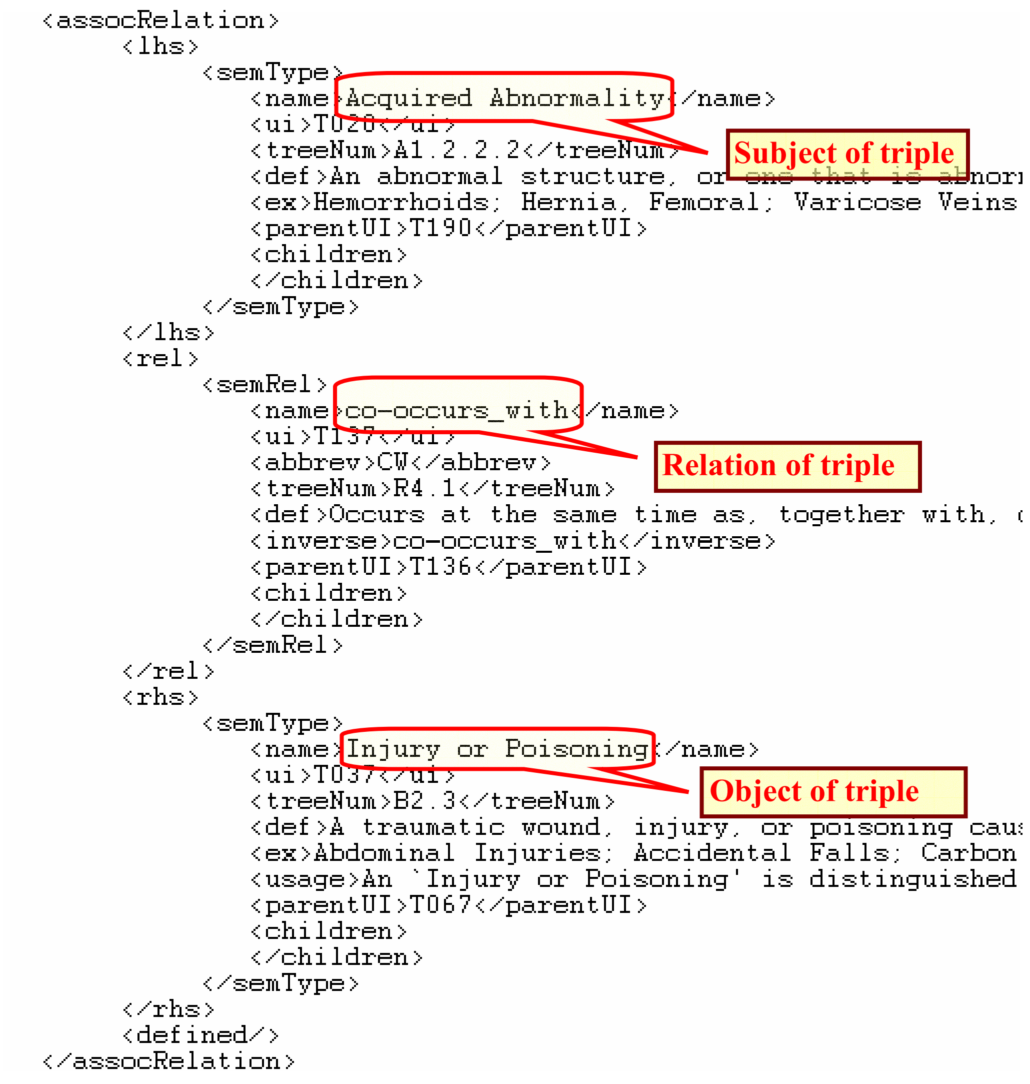


Figure 3: Architecture of the Semantic Browser Application

3.1. Ontology Used In The Process

To build the schema layer of the ontology used in our project, we have parsed the UMLS [1] dataset by using SAX java parser [33]. The UMLS dataset consists of several XML files.



```
<assocRelation>
     <lhs>
          <semType>
             <name>Acquired Abnormality</name>
             <ui>T020</ui>
             <treeNum>A1.2.2.2</treeNum>
             <def>An abnormal structure, or one that is abnorm
             <ex>Hemorrhoids; Hernia, Femoral; Varicose Veins
             <parentUI>T190</parentUI>
             <children>
             </children>
          </semType>
     </lhs>
     <rel>
          <semRel>
             <name>co-occurs_with</name>
             <ui>T137</ui>
             <abbrev>CW</abbrev>
             <treeNum>R4.1</treeNum>
             <def>Occurs at the same time as, together with, o
             <inverse>co-occurs_with</inverse>
             <parentUI>T136</parentUI>
             <children>
             </children>
          </semRel>
     </rel>
     <rhs>
          <semType>
             <name>Injury or Poisoning</name>
             <ui>T037</ui>
             <treeNum>B2.3</treeNum>
             <def>A traumatic wound, injury, or poisoning caus
             <ex>Abdominal Injuries; Accidental Falls; Carbon
             <usage>An `Injury or Poisoning' is distinguished
             <parentUI>T067</parentUI>
             <children>
             </children>
          </semType>
     </rhs>
     <defined/>
</assocRelation>
```

Figure 4: A triple in XML format from the UMLS dataset

Triples (terms and relationships associated between them) were extracted and stored in an RDF file. In the RDF schema file generated, there are 135 classes and 49 relationships.

In order to build the instance layer of the ontology used in our project, we used SAX java parser [33] to parse the MeSH XML file. This XML file is 240 MB in size. It contains 21,945 MeSH terms, but does not contain relationships between these terms. Because MeSH terms are represented as instances in our ontology, we have 21,945 instances in the instance layer of our ontology. From this XML file for each instance, we extracted the UI number which is a unique number for each MeSH term. We extracted the UMLS types of MeSH terms. As mentioned above, these UMLS types are represented as schema classes in the schema layer of our ontology.

The main benefit of this ontology is to let the user see the class of an entity instance, and all of the relationships that class has. By selecting any existing relationships, the user can traverse to another document which is indexed under the type that relationship is connected to.

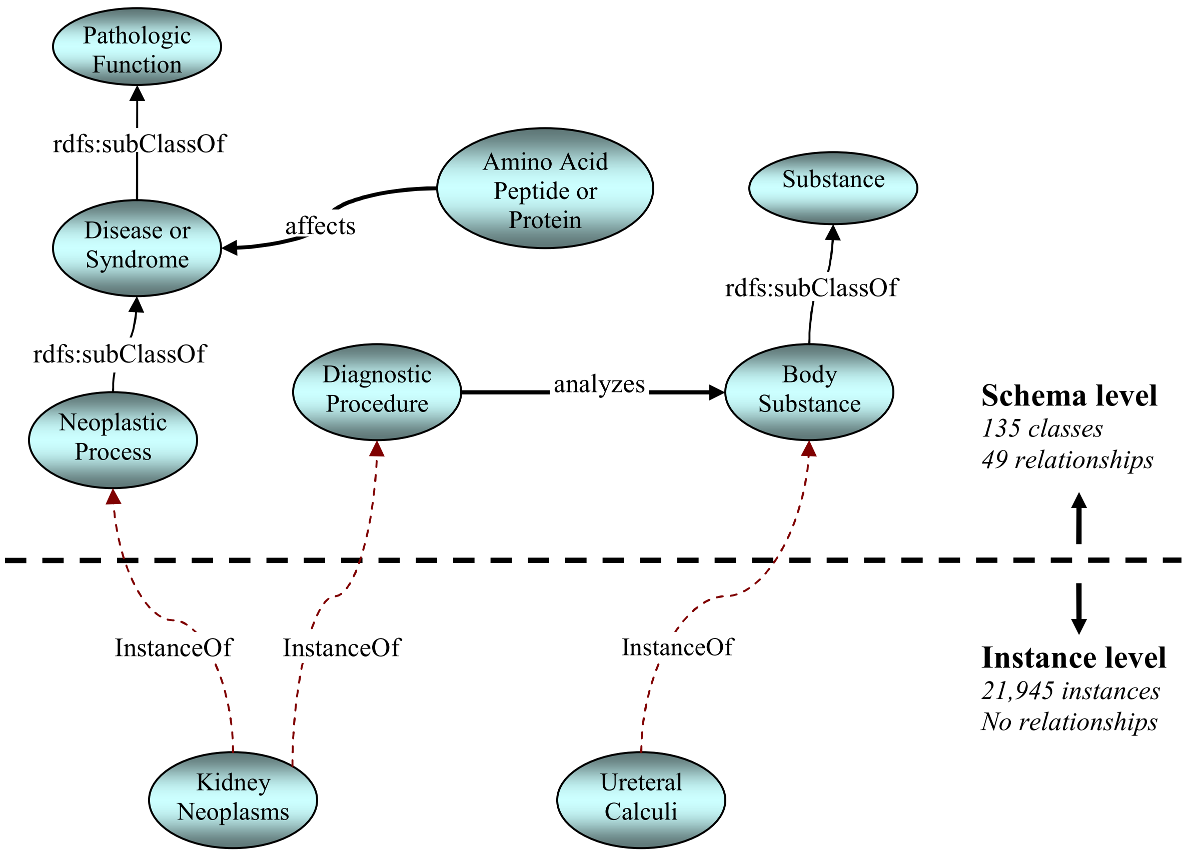


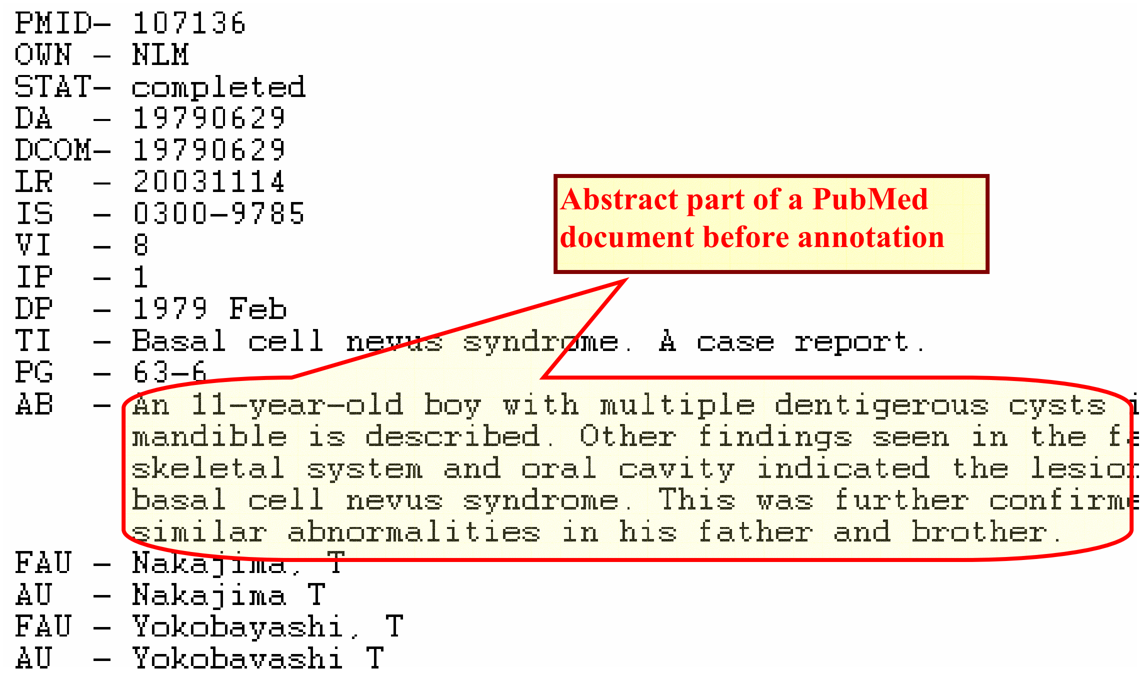
Figure 5: Schema level and instance level of the ontology used

Figure 5 illustrates the schema level and instance level of the ontology we used in our implementation. There is no direct relationship between concepts at the instance level. Without using the schema level, we can not say if there is any relation between “Kidney Neoplasm” and

"Ureteral Calculi". As can be seen from the Figure 5, Kidney Neoplasm has two types in the ontology schema. Ureteral Calculi, however, has only one type in the ontology schema. Figure 5 demonstrates that there is an "analyzes" relationship between "Diagnostic Procedure" and "Body Substance" at the schema level. Diagnostic Procedure is one of the types of Kidney Neoplasm. Body Substance is the type of Ureteral Calculi. Because we have the "analyzes" relationship between "Diagnostic Procedure" and "Body Substance" at the schema level, we conclude that there is also "analyzes" relationship between "Kidney Neoplasm" and "Ureteral Calculi" instances at the instance level.

3.2 Dataset Used In The Project

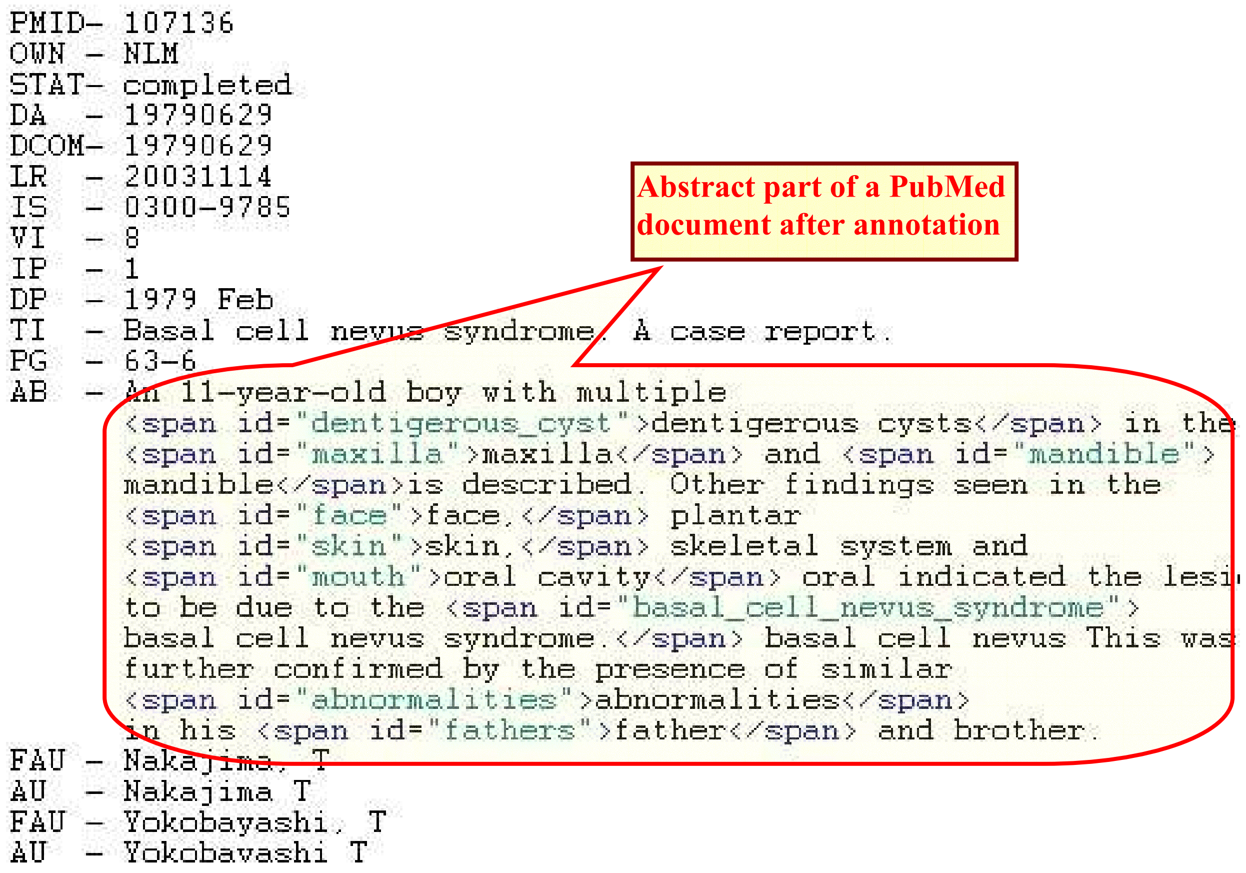
In this project, we use PubMed as a dataset [6]. It has 16 million documents with abstracts. We used a fraction of that dataset (48,252 documents). Each file is in the form of Figure 6.



```
PMID- 107136
OWN - NLM
STAT- completed
DA  - 19790629
DCOM- 19790629
LR  - 20031114
IS  - 0300-9785
VI  - 8
IP  - 1
DP  - 1979 Feb
TI  - Basal cell nevus syndrome. A case report.
PG  - 63-6
AB  - An 11-year-old boy with multiple dentigerous cysts i
      mandible is described. Other findings seen in the f
      skeletal system and oral cavity indicated the lesio
      basal cell nevus syndrome. This was further confirm
      similar abnormalities in his father and brother.
FAU - Nakajima, T
AU  - Nakajima T
FAU - Yokobayashi, T
AU  - Yokobayashi T
```

Figure 6: File format of PubMed dataset before annotation

To distinguish the MeSH terms in an abstract, those MeSH terms needed to be annotated. We have developed an algorithm for this annotation which works in linear time. The algorithm finds the MeSH terms in the sentences, and encloses them between specials tags. Figure 7 below is the annotated form of the abstract at Figure 6.



```
PMID- 107136
OWN - NLM
STAT- completed
DA  - 19790629
DCOM- 19790629
LR  - 20031114
IS  - 0300-9785
VI  - 8
IP  - 1
DP  - 1979 Feb
TI  - Basal cell nevus syndrome. A case report.
PG  - 63-6
AB  - An 11-year-old boy with multiple
      <span id="dentigerous_cyst">dentigerous cysts</span> in the
      <span id="maxilla">maxilla</span> and <span id="mandible">
      mandible</span>is described. Other findings seen in the
      <span id="face">face,</span> plantar
      <span id="skin">skin,</span> skeletal system and
      <span id="mouth">oral cavity</span> oral indicated the lesi
      to be due to the <span id="basal_cell_nevus_syndrome">
      basal cell nevus syndrome.</span> basal cell nevus This was
      further confirmed by the presence of similar
      <span id="abnormalities">abnormalities</span>
      in his <span id="fathers">father</span> and brother.
FAU - Nakajima, T
AU  - Nakajima T
FAU - Yokobayashi, T
AU  - Yokobavashi T
```

Figure 7: File format of PubMed dataset after annotation

The algorithm tokenizes the text, and processes it word by word. When the algorithm finds a word in the text and that term exist also in the ontology, it does not annotate that word immediately. Instead, the algorithm continues processing the words in the text and tries to find the term with maximum number of words in text which also exists in the ontology. The word

"LongestTerm" in the Algorithm Pseudocode below refers to the term with maximum number of words.

<u>Algorithm Pseudocode;</u>

```
AnnotateText(text)
1.  Tokenize the text into words
2.  For i = 1 to  #of words  Do {
3.      longestTerm = word[i]
4.      while(longestTerm is in MeSH terms){
5.          if(longestTerm + next_term  is in MeSH terms ){
6.              longestTerm = longestTerm + next_term
7.          } // End If
8.      } // End While
9.      if(longestTerm is in MeSH terms){
10.         AnnotateTerm(longestTerm);
11.     } // End If
12. } // End For
13. Return annotated text;
```

AnnotateTerm(text)

1. **text** = <span id=**text**>**text**</span>
2. Return text;

The term "zygote" and "zygote intrafallopian transfer" are in the ontology. A sentence which contains "zygote intrafallopian transfer" is in the abstract. The algorithm looks at the terms word by word. When it finds "zygote" in the sentence, it does not annotate it right away, but keeps continuing. The next word is “intrafallopian”. By combining these two words, we have “zygote intrafallopian”. We do not have “zygote intrafallopian” in our ontology, but we have some other terms which begin with “zygote intrafallopian”. We take the third word. If this third word is “transfer”, that means now we have “zygote intrafallopian transfer”. Also we have “zygote intrafallopian transfer” in our ontology. We do not annotate this three-word MeSH term, just because it exists in the ontology. We look at the ontology; if there is no four-word MeSH term in the ontology which begins with “zygote intrafallopian transfer”, then without needing to look at fourth word in the sentence we annotate this three-word MeSH term. What if the word after the “zygote intrafallopian” is not “transfer”? Then the algorithm goes back and annotates the first word “zygote”, because we kept record that “zygote” is contained in our ontology. After annotating it, we do not return where we were. But instead, we continue from “intrafallopian”. Because we have not checked if “intrafallopian” is contained in our ontology, or even if we do not have “intrafallopian”, still we may have some MeSH terms which begin with “intrafallopian”. Starting from “intrafallopian”, the algorithm applies the same process explained above.

3.3 Indexing The Articles In The Dataset

To index the documents in our dataset, we used Lucene [9]. To index the documents, by using the regular expression methods of the Java API (java.util.regex), we picked each annotated term in the documents, and associate each of these annotated terms with the document. As each document is associated with several MeSH terms, also each MeSH term is associated with several documents. To retrieve the documents associated with a particular MeSH term, we also use one of the methods of Lucene.

In our implementation, we used 21,945 MeSH terms and their synonyms. Thus, we used around 104,000 terms to index our dataset.

We have chosen Lucene, because it is written entirely in Java, and it is very commonly used open source project.

3.4 User Interface

The user interface is a simple HTML page using JavaScript and Cascading Style Sheet (CSS). JavaScript contains the AJAX engine [11]. AJAX engine serves an intermediary between the JavaScript methods and server. Because of its high performance and quick response time, we have used AJAX engine to send requests and to receive responses from the server. See Appendix-A for more details about AJAX.

3.5 Server-side Components

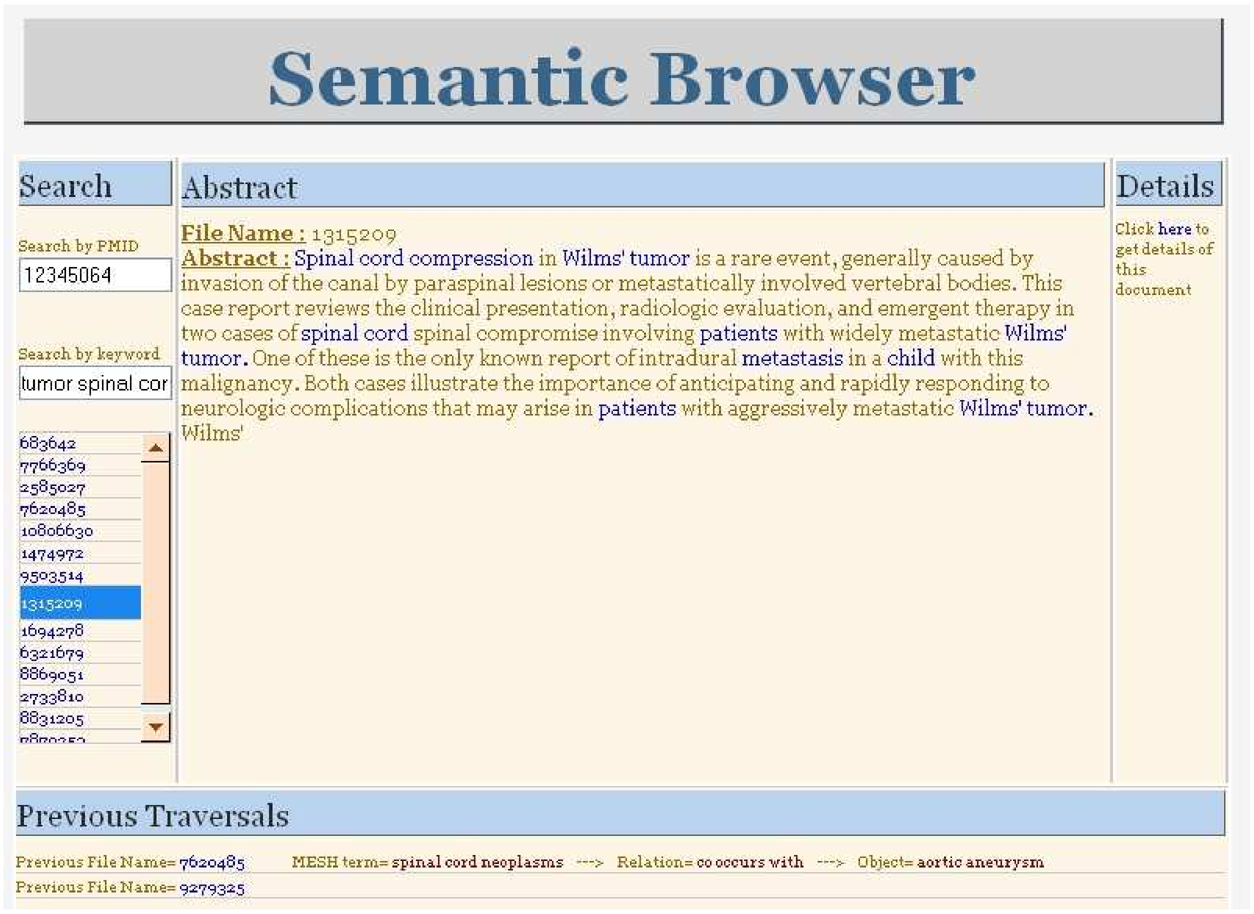
Server-side code is written in Java Server Pages (JSP) codes. JSP receives requests from the AJAX engine. When requests are sent from AJAX, instead of using GET, the POST method is used. Because the parameters are not carried by the URL, the URL remains clean during the navigation process. To use the AJAX technology, an instance of XMLHttpRequest object is created. After JSP receives the request, it parses the Request String to get all of the parameters coming from AJAX engine. Based on the parameters received, a JSP method is called. The method then calls one of the methods in the Java class. Java class is another component of the system architecture. It includes Lucene [9] methods, SemDis API [10][Appendix-B] methods, and some other methods used for annotation, or building the suggestion dropdown menus at the search boxes. After the JSP method calls one of the Java methods, the java method returns an array list (java.util.ArrayList) to the JSP. The JSP method then builds an XML format, and returns the response to the AJAX as in XML format. Because the response is in XML format, the

AJAX engine can receive the response by using “responseXML”. The methods in the JavaScript can easily manipulate the response by using XML DOM methods. We then insert the response received from the AJAX to inside of a table in the HTML file, without having to reload the whole HTML page.

To run the application on our server, we used Apache Tomcat [8] (Appendix-C) as the servlet container. To deploy and install the Java Server Pages to the Apache Tomcat, we used Apache Ant [7] (Appendix-D).

4. SEMANTIC BROWSER APPLICATION

The user interface is simply an HTML page. The articles to be shown in the user interface are annotated beforehand.



12345064

tumor spinal cor

683642
7766369
2585027
7620485
10806630
1474972
9503514
1315209
1694278
6321679
8869051
2733810
8831205

File Name : 1315209
Abstract : Spinal cord compression in Wilms' tumor is a rare event, generally caused by invasion of the canal by paraspinal lesions or metastatically involved vertebral bodies. This case report reviews the clinical presentation, radiologic evaluation, and emergent therapy in two cases of spinal cord spinal compromise involving patients with widely metastatic Wilms' tumor. One of these is the only known report of intradural metastasis in a child with this malignancy. Both cases illustrate the importance of anticipating and rapidly responding to neurologic complications that may arise in patients with aggressively metastatic Wilms' tumor. Wilms'

Click here to get details of this document

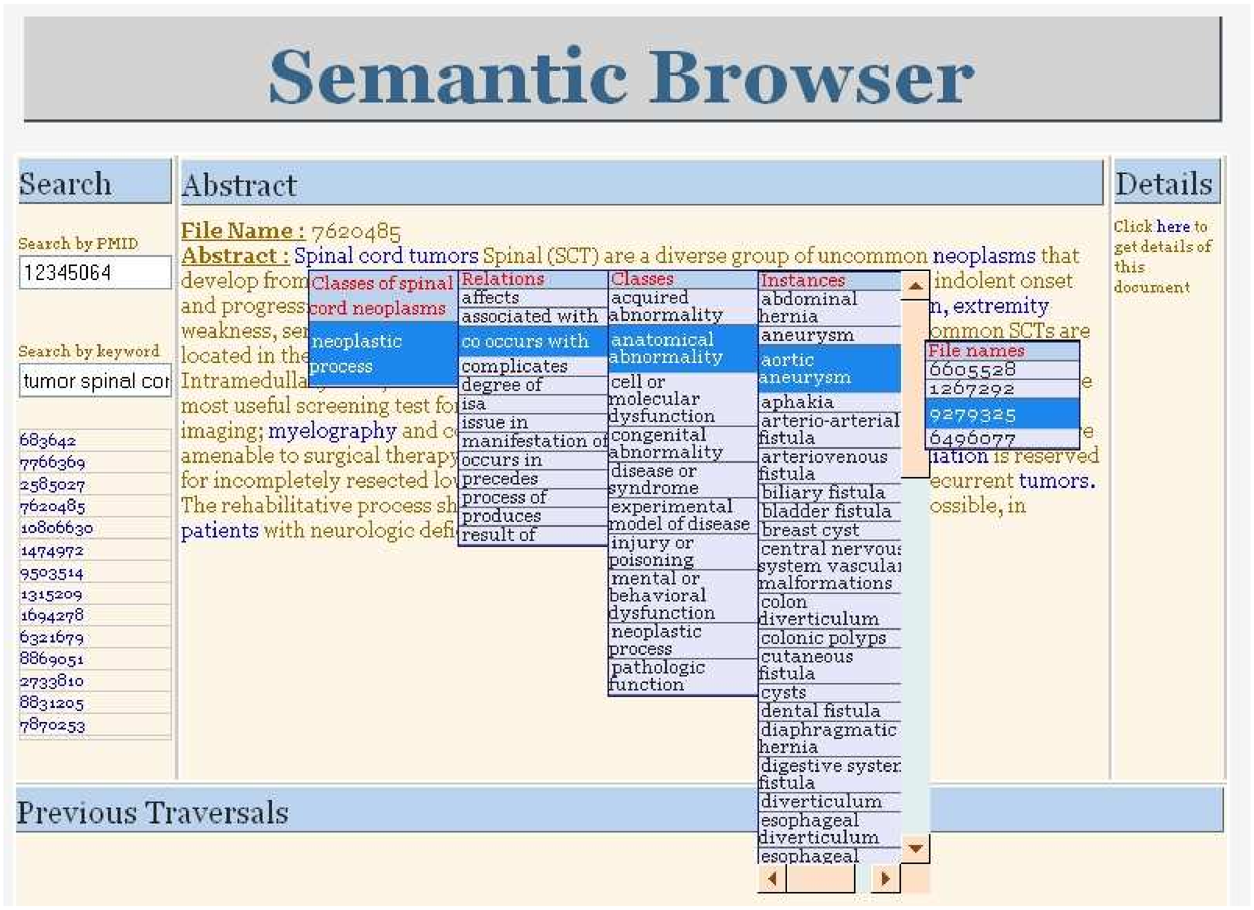
Previous File Name= 7620485 MESH term= spinal cord neoplasms ---> Relation= co occurs with ---> Object= aortic aneurysm
Previous File Name= 9279325

Figure 8: A screenshot of Semantic Browser tool

To begin traversing between documents, the user needs a document to begin with. On the left column of the user interface, there are two text boxes, where the user can search documents by either the unique PMID number of the article or by any MeSH term which may appear in the documents. As the user types in the textbox, just below the textbox, a popup menu appears showing the MeSH terms, which begin with the typed letters. Therefore, the size of the menu gets shorter as the user keeps typing. For this menu, a sorted set menu (java.util.SortedSet) is used. To generate a subset of the all MeSH terms (and their synonyms), the main sorted set needs to be built first. In order to build the subset menu when the user begins typing in the textbox, the main set has to be generated and loaded into memory first. The main sorted set which includes all MeSH terms (and their synonyms) is loaded by using a method from the Java file. The method of the Java file is called from the "onLoad" function of the HTML body, which is a Java Script event handler. Thus, as soon as the user visits the Web page, the sorted set begins to be generated from a text file, and loaded into main memory. If we did not generate the main set at onLoad function, when the user types a MeSH term in the search box, building the first subset menu would take some time to generate the main set first before generating the subset of the main set. Due to this latency, the user would not know that a popup menu is going to appear soon.

Just after loading the main set into memory, the "onLoad" function of the HTML body also calls the method in the Java file to load the ontology files into memory. When the user is interested in reading documents about any MeSH term (or its synonyms) from the popup menu, then the user has options to select the MeSH term from the menu by either clicking on it with mouse, or can navigate among the items by using arrow keys from the keyboard. After selecting any MeSH term from the list, the names of the files in which the selected MeSH term, or its synonym appears. For instance, by entering the keyword "neoplasm" in the search box, the user also gets file names in which "serum" appears, but "neoplasm" does not. Because, serum is a synonym of "neoplasm", the user does not miss that document which may be very important for the user. Those documents are returned, because when we index the documents with Lucene, we have also used the synonyms of the MeSH terms. While the number of MeSH terms is 21,945 , the number of terms used to index documents are around 104,000 including the synonyms of MeSH terms. That is, we have an average of four synonyms for each MeSH term.

The named entities in the text have different colors so that the user can understand that those terms are annotated based on some ontology, and will let the user traverse to another document. This traversal is based on the semantic relationships between the two documents. By simply hovering the mouse pointer on that named entity, a popup menu appears with a list of types under which that entity instance is placed. By hovering over any of those types, the user also gets the list of relationships of that type. By hovering over any of those relations from the list, the user gets the list of types that the relationship selected is connected to. By hovering over any of the types from the list, user gets a list of MeSH terms which are instance of hovered types. By hovering over any of the MeSH terms from the list, user gets a list of file names in which the MeSH term appears. Then, the user simply clicks on any file name in the menu and the abstract part of the selected file appears on the webpage.

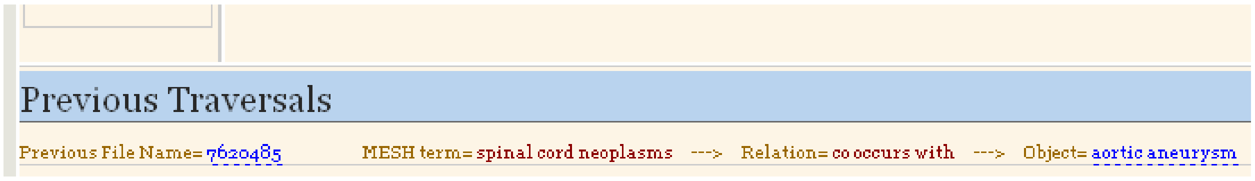


File Name : 7620485
Abstract : Spinal cord tumors Spinal (SCT) are a diverse group of uncommon neoplasms that
develop from … indolent onset
and progress… n, extremity
weakness, se… ommon SCTs are
located in the…
Intramedulla… e
most useful screening test fo…
imaging; myelography and c… e
amenable to surgical therapy… iation is reserved
for incompletely resected lo… ecurrent tumors.
The rehabilitative process sh… ossible, in
patients with neurologic defi…

Figure 9: A screenshot of Semantic Browser tool

The AJAX technology makes this process is very fast, because the page does not have to be reloaded every time the content of the popup menus is retrieved from the server, and abstract of the file is retrieved from the PubMed dataset.

By picking a relationship in the popup menu, the user can go to another document. Just after this process, the name of the previous document is written into the “Previous Traversals” section of the Semantic Browser tool. Just next to the name of the file, the triple that the user went through in that document is written. For instance, let us assume that the user reads the abstract of the document named “7620485”. By hovering over the term “spinal cords neoplasm”, the user picks the relationship “co occurs with”. Then, the user picks “aortic aneurysm”. Then among the file names appearing in the popup menu, the user selects the document "9279325". Just at the end of this process, the name of the previous file, 7620485, is written into "Previous Traversals" section. The traversal done in that document is written next to the file name, as shown in Figure 10.

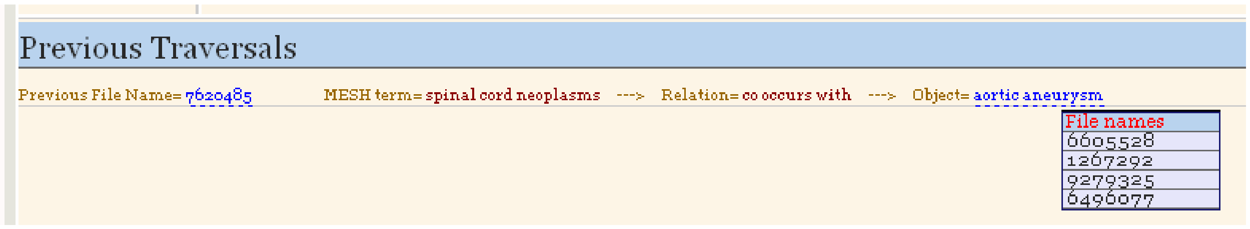


Previous Traversals

Previous File Name= 7620485 MESH term= spinal cord neoplasms ---> Relation= co occurs with ---> Object= aortic aneurysm

Figure 10: Previous Traversals section of Semantic Browser.

By simply clicking on the file name, the user can go to previous document easily. If the user does not like the current document and wants to read other documents that contain the term "aortic aneurysm", without needing to go back to previous file, user can hover on the term "aortic aneurysm" in the "Previous Traversals" section (See Figure 11).



Previous Traversals

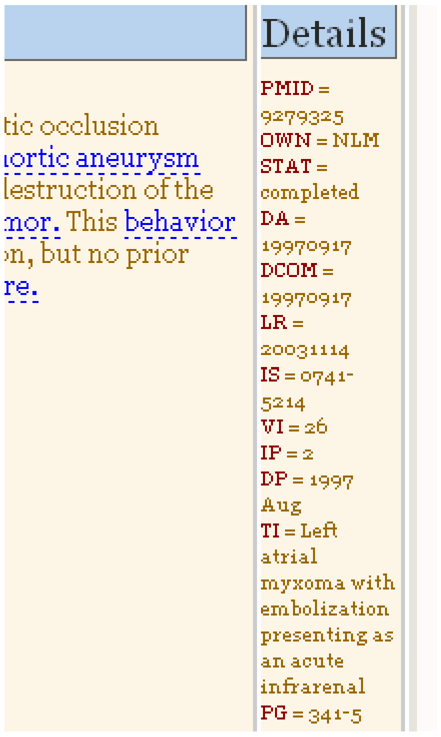
Previous File Name= 7620485 MESH term= spinal cord neoplasms ---> Relation= co occurs with ---> Object= aortic aneurysm

File names
6605528
1267292
9279325
6496077

Figure 11: Previous Traversals section of Semantic Browser.

When the popup menu appears, the user can select another file name. The abstract of the document selected is written into "Abstract section" of the Semantic Browser. Such a history mechanism is meant to be more than just a browsing aid. These traversals that are recorded contain entities and named relationships. They can therefore serve as “Semantic” indices in the document corpus. If the user discovered new and interesting information during such a traversal she could choose to remember the trail that she followed to get the information. In the future we plan to build a relationship-based document index that will allow retrieval of documents that were found along such trails. This is an idea very similar to “trailblazing” described by Dr. Vannevar Bush [4].

The user can also see the details of the current document, by clicking on the “Details” section of the Semantic Browser tool.



tic occlusion
ıortic aneurysm
lestruction of the
mor. This behavior
ın, but no prior
re.

PMID = 9279325
OWN = NLM
STAT = completed
DA = 19970917
DCOM = 19970917
LR = 20031114
IS = 0741-5214
VI = 26
IP = 2
DP = 1997 Aug
TI = Left atrial myxoma with embolization presenting as an acute infrarenal
PG = 341-5

Figure 12: Details section of the Semantic Browser.

5. RELATED WORK

In this thesis, we presented an implementation that classifies documents, builds a Semantic Web by using the semantics of the relationships, and enables users to navigate among the documents in a meaningful way by using the relationships as opposed to hyperlinks which do not carry any semantics of relationships between the linked documents. In this section, we discuss some of the related work to our approach.

There has been a considerable amount of work done to discover the relationships between terms in unstructured text by using natural language processing techniques. There are several NLP (Natural Language Processing) tools that are used by third parties to parse the sentences in their dataset to extract the relationships. GATE (General Architecture for Text Engineering)[14], CGPARSER[15], OpenNLP[16], and Link Grammar[17] are some of the commonly used NLP tools. In our implementation however, we used a structured text (XML file) to generate triples (two entities and the relationship between them). By using these triples, we generate our ontology.

There are many algorithms for automatic clustering, such as the K-Means algorithm [18], hierarchical clustering [19] and Expectation Maximization [20] to form the clusters. These algorithms do not utilize the relationships that exist between the terms. In their paper "Text Clustering using Semantics" [21], the authors have proven that using relationships between the documents increases the accuracy. In their work, they demonstrate that the text clustering using semantics outperforms other clustering algorithms which do not use semantics of relationships. The advantage of our application is that we have the relationships not from document to document, but from term to term. These terms may appear either in the same document, or also in different documents. Thus, any two documents may be connected to each other via multiple relationships that exist between term pairs.

There are some "Semantic Browser" implementations [22, 23, 24, 25, 26] that enable users to navigate between the different websites by using semantic relationships. To the best of our knowledge, our Semantic Browser tool is the only existing work that uses AJAX technology [11]. By using AJAX technology, and JavaScript that works on the client-side, users do not have to install anything to their machine. AJAX technology works on almost all of the commonly-used browsers, such as; Microsoft Internet Explorer version 5.0 and above, Mozilla Firefox, SeaMonkey, Camino, Flock, Epiphany, Galeon and Netscape version 7.1 and above. Apple Safari version 1.2 and above.

Anyone who has used Flickr, GMail, Google Suggest, or Google Maps will realize that a new breed of dynamic web applications is emerging. These applications look and act very similar to traditional desktop applications without relying on plug-ins or browser-specific features. Web applications have traditionally been a set of HTML pages that must be reloaded to change any portion of the content. Technologies such as JavaScript programming language and cascading style sheets (CSS) have matured to the point where they can be used effectively to create very dynamic web applications that will work on all of the major browsers [27]. Likewise our implementation, although being a web application, looks and acts like a desktop application.

Semantic Web content and Semantic Web tools depend each other. Without sufficient Semantic Web content, few tools will be written to consume it; without many such tools, there is little appeal to publish Semantic Web content. In the Piggy Bank project (MIT) [26], they define this problem as "chicken-and-egg problem", and they provide a web browser extension called Piggy Bank that lets users make use of Semantic Web content within web content as users browse the Web. By using Piggy Bank, a tool integrated into the contemporary web browser, Firefox, web users extract information items from within web pages and save them in the Semantic Web format RDF [30]. When the user visits a web site, if there is a structured data of the same information available to retrieve, the web browser shows a “data coin” icon in the status bar for each site. By clicking on that icon, the “pure” information from each web site is collected. The browser Piggy Bank shows the information items it has collected from one of the sites, right inside the same browser window.

The user can also tag an item with one or more keywords, to help him/her find it later. The “tag completion” dropdown suggests previously used tags that the user can pick from. The user can also tag or save several items together.

To make this effort in collaborative way, with one click on the “Publish” button for each item, the user publishes information to the Semantic Bank. Thus, other user can use this information.

The users generate metadata, and attach to information very easily. Although having some similarity to our project, their approach differs from ours in the usage of semantics of relationships. They cluster documents by using metadata in RDF, but not using relationships.

The project “Magpie” [32] is similar to our project. However, in Magpie, a plug-in is needed to be installed to browser, whereas in our project no plug-in is needed to install. In the Magpie, they have very few relationships (around 6-7), whereas we have 49 relationships in our project. Because they have also very few classes (around 4 usually), they show the different classes with different colors. Because we have 135 classes in our project, we could not show them in different colors.

6. CONCLUSIONS AND FUTURE WORK

In this thesis, we described a tool that allows user to navigate between documents using virtual relationships. However, currently we use relationships that exist between the types of instances, not between the instances. This method works most of the time. However, we can not assume that it holds true all the time. Because it is not guaranteed that when there is a relationship between any two types, the relationship should exists between every instances of these two types. For instance; in our ontology schema, there is a relationship "adjacent_to" between classes "body part organ" and "body location or region". Some of the instances of the class "body part organ" are breast, prostate. Some of the instances of the class "body location or region" are cheek, chin, elbow, and abdomen. If we try to put "adjacent_to" relationship between some of instances, we would get incorrect statements. If we put "adjacent_to" relationship between prostate and chin, then this would be correct statement. In order to always get correct results, we need relationships at the instance level of the ontology, not at the schema level. An ontology that has relationships at the instance level has been created [29]. In a near future, we will integrate this ontology into our Semantic Browser tool. Once we integrate that ontology into our tool, then this tool will be used also as a means to evaluate that work.

Currently, the indexing using Lucene [9] does not take relationships into consideration. We will index our dataset by also using the relationships that are extracted from the documents. In this way, more relevant documents will be returned to the user.

Besides providing the users ease of browsing between the documents, our goal is also to reduce the time that a user has to spend with documents. Among the returned multiple documents, we should rank the documents based on their relevance to the triples, the user may be interested to learn. As the ranking mechanism, we plan to use the method explained in Boanerges Aleman-Meza’s paper “Searching and Ranking Documents based on Semantic Relationships” [28].

Our work is one of the earliest attempts at utilizing the semantic relationships to support browsing and navigation of a document space.

To the best of our knowledge, this tool is the only existing Semantic Browser tool that uses AJAX technology. Therefore, it works in any contemporary web browser.

7. REFERENCES

[1] Unified Medical Language System. http://umlsinfo.nlm.nih.gov

[2] Medical Subject Headings. http://www.nlm.nih.gov/mesh

[3] Home Remedies and Natural Cures for Common Illnesses http://www.home-remedies-for-you.com/remedy/Measles.html

[4] Vannevar Bush: As We May Think. The Atlantic Monthly 176(1): 101-108 (1945)

[5] Kshitij Shah, Amit P. Sheth: Logical Information Modeling of Web-Accessible Heterogeneous Digital Assets. 266-275

[6] National Center for Biotechnology Information. http://www.ncbi.nlm.nih.gov

[7] The Apache Ant Project. http://ant.apache.org

[8] Apache Tomcat Home Page. http://tomcat.apache.org

[9] Apache Lucene Home Page. http://lucene.apache.org/java/docs

[10] Semantic Discovery: Discovering Complex Relationships in Semantic Web. http://lsdis.cs.uga.edu/projects/semdis/sweto/index.php?page=5

[11] Ajax: A New Approach to Web Applications by Jesse James Garrett. http://www.adaptivepath.com/publications/essays/archives/000385.php

[12] Ajax. http://en.wikipedia.org/wiki/AJAX

[13] A Simpler Ajax Path, by Matthew Eernisse. 05/19/2005. http://www.onlamp.com/pub/a/onlamp/2005/05/19/xmlhttprequest.html

[14] GATE, General Architecture for Text Engineering. http://gate.ac.uk

[15] Contextual Grammars. http://www.uni-koblenz.de/~harbusch/CG-PARSER/welcome-cg.html

[16] OpenNLP Home Page. http://opennlp.sourceforge.net

[17] Link Grammar. Davy Temperley, Daniel Sleator, John Lafferty. http://www.link.cs.cmu.edu/link

[18] J. B. MacQueen (1967): "Some Methods for classification and Analysis of Multivariate Observations, Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability", Berkeley, University of California Press, 1:281-297

[19] A.K. Jain and R.C. Dubes. Algorithms for Clustering Data, Prentice Hall, Englewood Cliffs NJ, U.S.A., 1988.

[20] Arthur Dempster, Nan Laird, and Donald Rubin. "Maximum likelihood from incomplete data via the EM algorithm". Journal of the Royal Statistical Society, Series B, 39(1):1–38, 1977

[21] Choudhary, R., Bhattacharyya, P.: Text Clustering Using Semantics. The 11th International World Wide Web Conference, WWW2002, Honolulu, Hawaii, USA (2002)

[22] Haystack Project. http://haystack.lcs.mit.edu

[23] Semantic Browser. Associative Similarity Browsing. http://semanticbrowser.aspasia-systems.de

[24] BigBlogZoo Home Page. http://www.bigblogzoo.com

[25] Amblit Navigator. http://www.amblit.com/products

[26] Piggy Bank Home Page. http://simile.mit.edu/piggy-bank

[27] AJAX. http://java.sun.com/developer/technicalArticles/J2EE/AJAX

[28] Searching and Ranking Documents based on Semantic Relationships, Boanerges Aleman-Meza, ICDE'06 PhD Workshop, April 3, 2006

[29] A Framework for Schema-Driven Relationship Discovery from Unstructured Text. Cartic Ramakrishnan, Krys J. Kochut and Amit P. Sheth. Submitted to ISWC2006.

[30] Resource Description Framework. http://www.w3.org/RDF

[31] Semantic Browser. http://lsdis.cs.uga.edu/projects/semdis/SemanticBrowser/

[32] Martin Dzbor, John B. Domingue, and Enrico Motta. Magpie - towards a semantic web browser. In Proceedings of the 2nd Intl. Semantic Web Conference,October 2003. Sanibel Island, Florida.

[33] Java SAX Parser.
http://java.sun.com/j2se/1.4.2/docs/api/javax/xml/parsers/SAXParser.html

APPENDICES

APPENDIX A - AJAX (Asynchronous JavaScript And XML)

Ajax, shorthand for Asynchronous JavaScript and XML, is a Web development technique for creating interactive web applications. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user makes a change. This is meant to increase the web page's interactivity, speed, and usability.

The term "Ajax" was coined by Jesse James Garrett in his February 2005 article Ajax: A New Approach to Web Applications [11].

Some websites use only Java Script, without AJAX engine in it. In such websites, when the user clicks on some special terms, or hover the mouse over those terms in the website, some additional information appears in no time. The behavior described here looks good, but the problem is that all of that information needs to be preloaded to the client side. After all those information, including the ones that even may not be clicked, are also loaded into the client side. By clicking or hovering over them, we simply turn the hidden status of the related information into visible mode. AJAX architecture is a great solution for this problem. By using the AJAX, instead of loading all of the webpage content into client side, by clicking, or hovering over a term, we make request to the server using the clicked term only as an input, and get the response from server only related to that term, and load the result to the website at client side dynamically. Preloading all the information might be scalable for small amount of data, however, it would not be okay for our case, where we have thousands of named entities that we will be using to request the server, and extract the related information from the ontology.

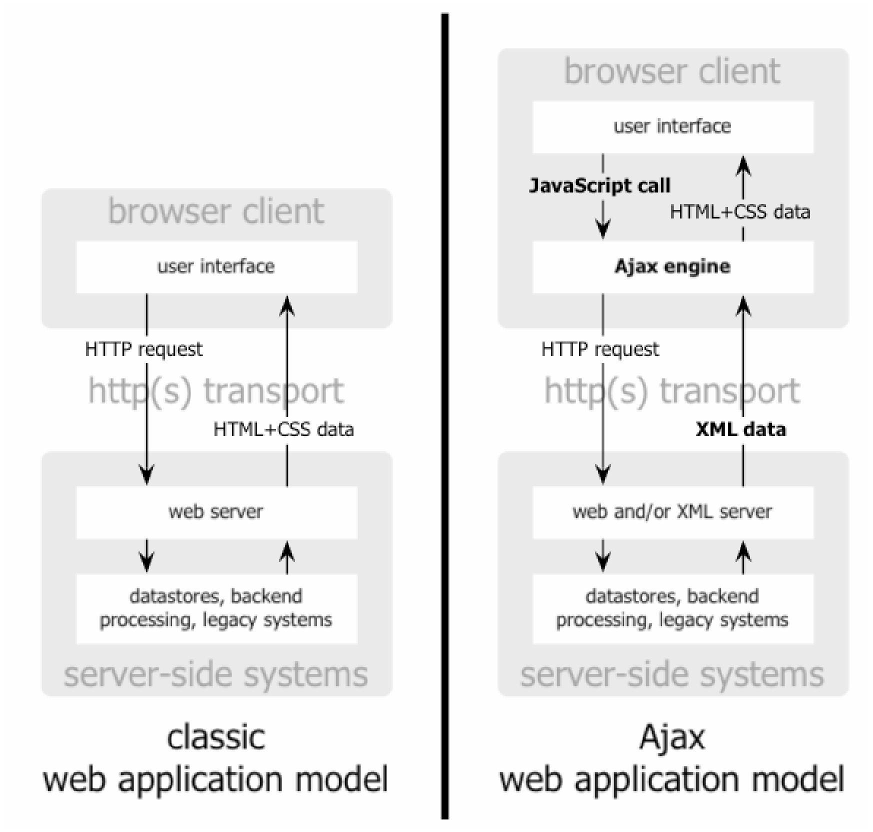


Figure 13: The traditional model for web applications (left) and the Ajax model (right) [11].

Not having to reload the whole page, but just insert the data returned from the server into the webpage boosts the speed considerably. Thus, rather than a web application, it feels like desktop application due to its speed, as we have experienced the use of Ajax at the Gmail, Google Maps, Google suggest, etc.

Browsers that support Ajax are as follows; Microsoft Internet Explorer version 5.0 and above, Mozilla Firefox, SeaMonkey, Camino, Flock, Epiphany, Galeon and Netscape version 7.1 and above. Apple Safari version 1.2 and above. Opera browsers version 8.0 and above, including Opera Mobile Browser version 8.0 and above [12].

Despite its name, you can use the XMLHttpRequest object with more than just XML. You can use it to request or send any kind of data--keep in mind, though, that JavaScript processes the response from the server [13].

While no browser plug-in is required for Ajax, it requires users to have JavaScript enabled in their browsers. This applies to all browsers that support Ajax except for Microsoft Internet Explorer 6 and below which additionally require ActiveX to be enabled, as the XMLHTTP object is implemented with ActiveX in this browser. Internet Explorer 7, however, will implement this interface as a native JavaScript object and hence does not need ActiveX to be enabled for Ajax to work [12].

APPENDIX B – SemDis API

In the LSDIS Lab, we have investigated about loading big RDF files. Brahms is a RDF datastore designed for that purpose. Various aspects of BRAHMS RDF Store influenced the design of SemDis API. In fact, the underlying UML model is the same. Thus, this Java implementation can be used for quick prototyping and later ported into C++ Brahms implementation whenever loading large RDF datasets is required. Alternatively, the Java bindings of Brahms can be used as well. Such java bindings were coded with respect to a JAVA API, basically, a set of java interfaces, which we call SemDis API.

There is also a Java-only main-memory implementation of this API, and that is what we have used in this Semantic Browser project.

APPENDIX C – Apache Tomcat

As the servlet container in our application, we used Apache Tomcat. Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and Java Server Pages technologies. The Java Servlet and Java Server Pages specifications are developed by Sun under the Java Community Process.

Apache Tomcat is developed in an open and participatory environment and released under the Apache Software License. Apache Tomcat is intended to be a collaboration of the best developers from around the world [8].

APPENDIX D – Apache Ant

We used Apache Ant for the compilation and installation of Java & JSP part. Apache's ANT is a java build tool that replaced using modifications of make early in the history of java programming. Most Modern IDE's either integrate with using ANT or have a plugin for using ANT as your Java Build Tool.

Jame Duncan Davidson started creating ANT in early 1999 to solve a problem that he had in developing Tomcat. He was using shell scripts to compile that were introducing errors in the build process. Thus he wrote the ANT tool to control the build process from an xml script and this tool became an Apache project in 2000.

Although ANT is a stand alone tool it has become the defacto standard java build tool both in stand alone forms and integrated in the top IDEs. Thus it is important for beginning java programmers to get a good grasp of how to use the ANT java build tool in their projects [7].