Traversing Documents by Using Semantic Relationships

Bilal Gonen[1]
[1]Computer Science and Engineering Department, University of Nevada, Reno, Reno, Nevada, U.S.A.

Abstract— *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.*

Keywords: semantic web, semantic browser, data mining

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 depends on whether the author of the text includes 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 is 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. 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

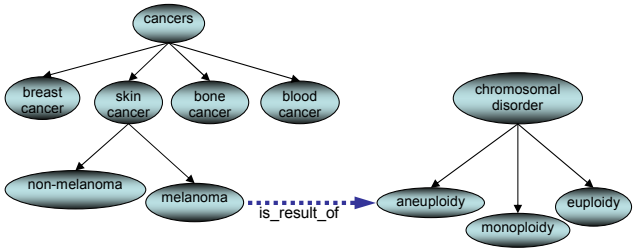

Fig. 1: Relationship in the ontology

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.

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 our paper, 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 no hyperlink between those documents.

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

In section 2, we describe the concept “Relationship Web”. Then, we explain the system architecture in Section 3. In that section, we describe the ontology and dataset used in the project. Section 4 shows the “Semantic Browser” tool [25]. In section 5, we review related work. In section 6, conclusions and future work are explained.

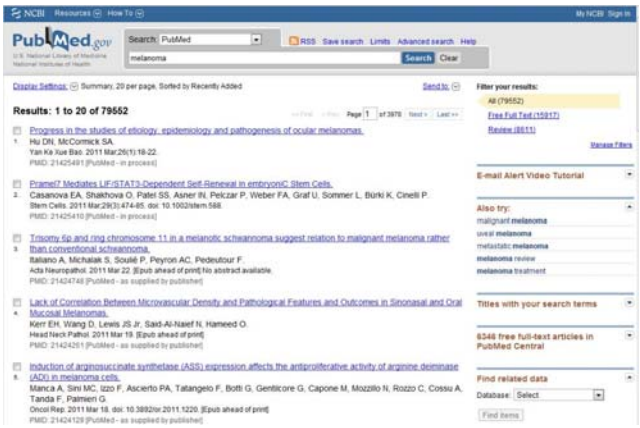

Fig. 2: A screenshot from PubMed Website

2. Building Relationship Web

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 websites have some system to suggest related articles to users. 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.

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 website uses a statistical mechanism. Such statistical mechanisms may bind two terms together as "related" just because 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.

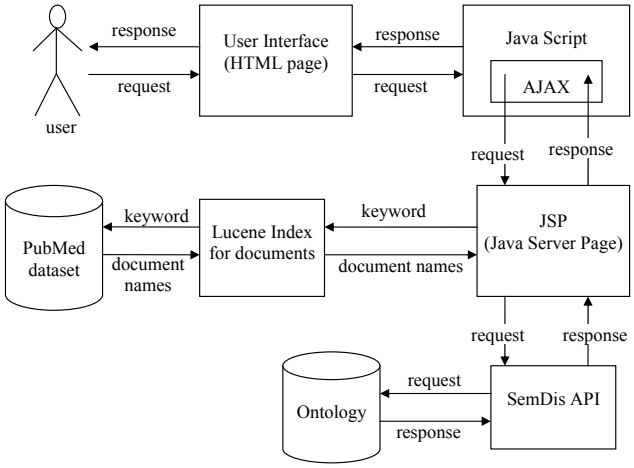

Fig. 3: Architecture of the Semantic Browser Application

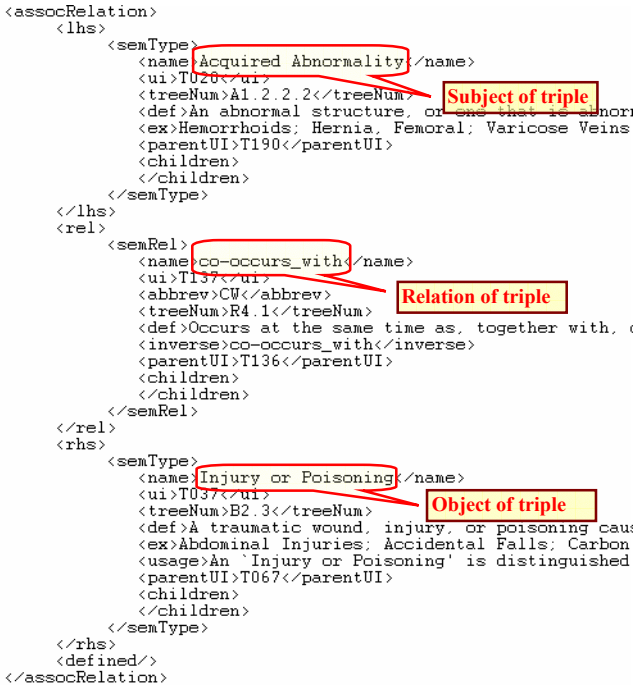

Fig. 4: A triple in XML format from the UMLS dataset

3. System Architecture

In this chapter, we explain the ontology and dataset used in the project.

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. The UMLS dataset consists of several XML files. 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 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

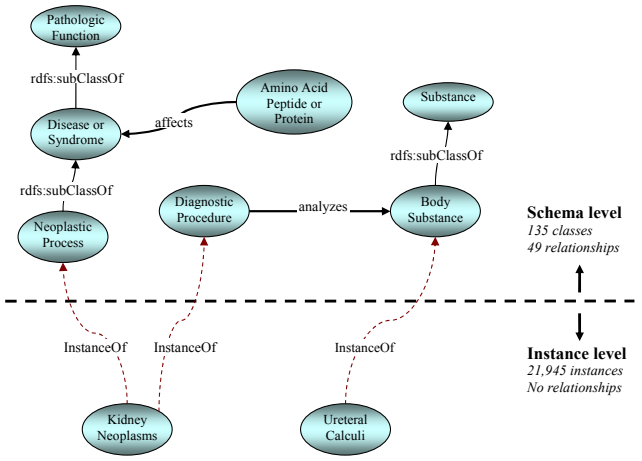

Fig. 5: Schema level and instance level of the ontology used

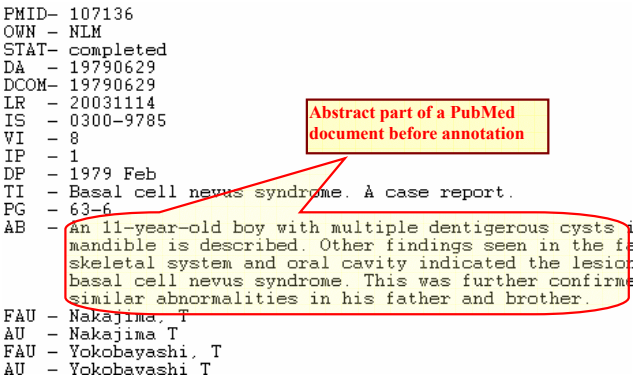

Fig. 6: File format of PubMed dataset before annotation

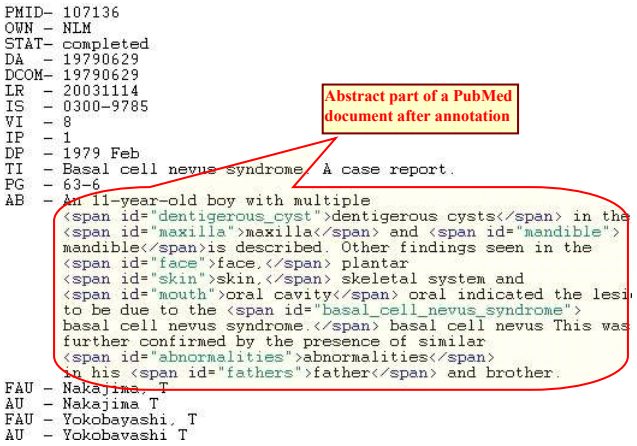

Fig. 7: File format of PubMed dataset after annotation

can traverse to another document which is indexed under the type that relationship is connected to.

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). Their format is illustrated in Figure 6.

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.

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 8 refers to the term with maximum number of words.

3.3 Processing Data And Indexing The Articles

To index the documents in our dataset, we used Lucene [7]. 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. In our implementation, we used 21,945 MeSH terms and their synonyms. Thus, we used around 104,000 terms to index our dataset.

The user interface is a simple HTML page using JavaScript and CSS. JavaScript contains the AJAX engine [9]. AJAX engine serves as 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. Server-side code is written in Java Server Pages (JSP) codes. After JSP receives the request from AJAX engine, it

```
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;
```

Fig. 8: Pseudocode of AnnotateText Algorithm

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 includes Lucene [7] and SemDis API [8] methods. Java methods return an array list (java.util.ArrayList) to the JSP. The JSP method returns the response to the AJAX as in XML format. 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 the HTML file, without having to reload the whole HTML page.

4. Semantic Browser Application

The articles to be shown in the user interface are annotated beforehand. 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 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.

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.

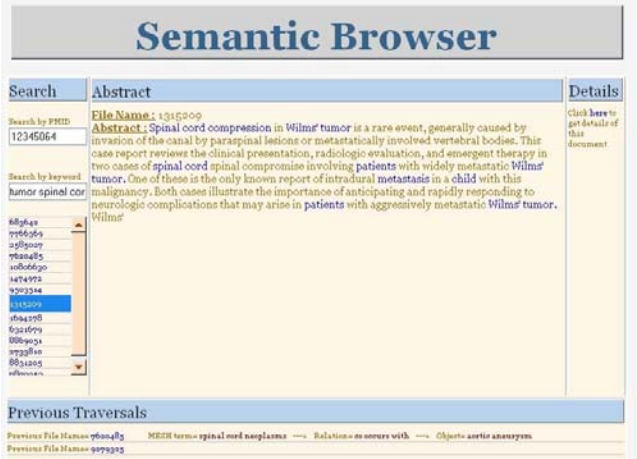

Fig. 9: A screenshot of Semantic Browser tool

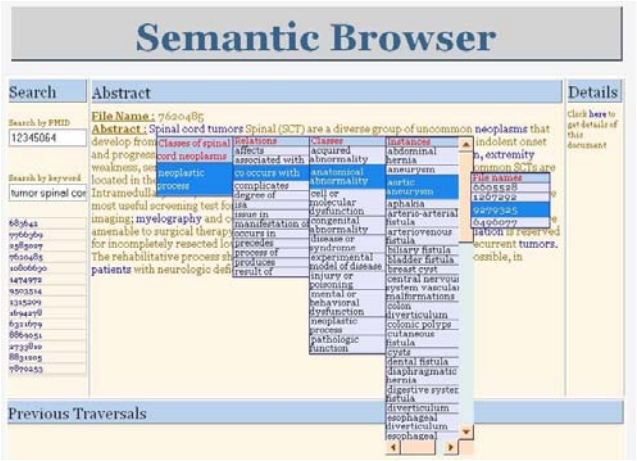

Fig. 10: A screenshot of Semantic Browser tool

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 other documents. 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.

The AJAX technology makes this process is very fast, because the page does not have to be reloaded every time

Fig. 11: Previous Traversals section of Semantic Browser

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. 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. By simply clicking on the file name, the user can go to previous document easily. If the user wants to read other documents that contain the term "aortic aneurysm", without having to go back to previous file, user can hover on the term "aortic aneurysm" in the "Previous Traversals" section. When the popup menu appears, the user can select another file name and go to that document (See Figure 11).

Such a history mechanism is meant to be more than just a browsing aid. These recorded traversals 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].

5. Related Work

In this paper, 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) [10], CGPARSER [11], OpenNLP [12], and Link Grammar [13] 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 [14], hierarchical clustering [15] and Expectation Maximization [16] to form the clusters. These algorithms do not utilize the relationships that exist between the terms. In their paper "Text Clustering using Semantics" [17], 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 [18]–[22] 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 [9]. 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, Mozilla Firefox, Chrome, SeaMonkey, Camino, Flock, Epiphany, Galeon, Netscape, and Apple Safari.

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 [23]. 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) [22], 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 RDF format [24]. 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. 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” [26] 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. 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. Having 135 classes in our project, this was not possible.

6. Conclusions and Future Work

In this paper, we described a tool that allows users 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 not be a correct statement. In order to always get correct results, we need relationships at the instance level of the ontology, not at the schema level.

Currently, the indexing using Lucene [7] 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.

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

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] Apache Lucene Home Page. http://lucene.apache.org/java/docs
[8] Semantic Discovery: Discovering Complex Relationships in Semantic Web. http://lsdis.cs.uga.edu/projects/semdis/sweto/index.php?page=5
[9] Ajax: A New Approach to Web Applications by Jesse James Garrett. http://www.adaptivepath.com/publications/essays/archives/000385.php
[10] GATE, General Architecture for Text Engineering. http://gate.ac.uk
[11] Contextual Grammars. http://www.uni-koblenz.de/ harbusch/CG-PARSER/welcome-cg.html
[12] OpenNLP Home Page. http://opennlp.sourceforge.net
[13] Link Grammar. Davy Temperley, Daniel Sleator, John Lafferty. http://www.link.cs.cmu.edu/link
[14] 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
[15] A.K. Jain and R.C. Dubes. Algorithms for Clustering Data, Prentice Hall, Englewood Cliffs NJ, U.S.A., 1988.
[16] 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):1Ű38, 1977
[17] Choudhary, R., Bhattacharyya, P.: Text Clustering Using Semantics. The 11th International World Wide Web Conference, WWW2002, Honolulu, Hawaii, USA (2002)
[18] Haystack Project. http://haystack.lcs.mit.edu
[19] Semantic Browser. Associative Similarity Browsing. http://semanticbrowser.aspasia-systems.de
[20] BigBlogZoo Home Page. http://www.bigblogzoo.com
[21] Amblit Navigator. http://www.amblit.com/products
[22] Piggy Bank Home Page. http://simile.mit.edu/piggy-bank
[23] AJAX. http://java.sun.com/developer/technicalArticles/J2EE/AJAX
[24] Resource Description Framework. http://www.w3.org/RDF
[25] Semantic Browser. http://lsdis.cs.uga.edu/projects/semdis/SemanticBrowser/
[26] 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.