Searching information sources based on the Semantic Web

Mark Ingram and Weiru Liu

School of Computer Science
Queen's University Belfast
Belfast, Co Antrim BT7 1NN, UK

Abstract. With the vision of the Semantic Web being grasped and realised more and more, the number of information sources that are available in the Semantic Web format is fast growing and will continue to do so in an exponential way. With this exponential growth in the number of information sources on the Semantic Web, it is necessary to be able to search through them automatically for desirable information. This paper presents a preliminary implementation of a semantic search engine for use on the Semantic Web. It makes use of an ontology mapping method that exploits the equivalence of concepts and examines the substring makeup of each subject, predicate and object in the source ontologies.

1 Introduction

1.1 The need for a search engine on the Semantic Web

No one doubts the tremendous success the World Wide Web (the Web) has enjoyed in recent years. One element of the Web's success is that it allows easy access to information on practically any topic. All a user has to do is navigate to a search engine with keywords indicating what is being searched. The search results contain links to web pages where the user's keywords appear. The Web's success has also brought it down a path with many stumbling blocks, one of them stems from the Web's worldwide utilisation. Just as translators or a common language are required for people from around the world to communicate effectively, so translators or a common language are required for effective communication and processing on the Web.

For example, in the UK a *postcode* is included in an address but in the USA the equivalent concept is called a *zipcode*. Still somewhere else the concept may be called an *area code*. If a user entered *postcode* into a search engine, the search engine would return a set of results with links to pages where the exact keyword *postcode* was found. The results set would not contain the pages where *zipcode* or *area code* were used in place of *postcode* . This is an example of keyword-based searching and is a method of searching that wholly disregards semantics.

The Semantic Web aims to add meaning to content stored on the Web. It is an extension of the current Web. Based on the Semantic Web, searching should be more intelligent. With the Semantic Web, ontologies can be defined that describe formally and explicitly what the concepts contained in a Web page mean. Axioms can be used explicitly to state the truths about information, such as concept *postcode* is equivalent

to concept *zipcode*. A search with keyword *postcode* based on the Semantic Web should return Web pages containing word *zipcode* too, since these two concepts are equivalent.

The first step is to develop a commonly acceptable ontology language for the Semantic Web.

1.2 Semantic Web languages

An ontology describes a domain of discourse in a formal way. An ontology is a finite collection of concepts that are fundamental in a domain of discourse. The ontology will also describe the relationships between these concepts.

An ontology consists of a set of concepts, C, that are arranged in a hierarchical fashion much like a taxonomy, H_C, with relationships existing between individual concepts, R_C. Relationships, too, can be arranged in a hierarchy, H_R . Instances of specific concepts, I, are interconnected by specific relationship instances, R_I. It is also possible to define axioms, A, that can be used to infer knowledge from existing statements. For an extended definition of this view of an ontology, please read [Stu03].

There are several commonly accepted ontology languages so far, such as, DAML+OIL, OWL, etc. They are all built on RDF and RDFS [MMM04]. RDF (Resource Description Framework) is an abstract data model for describing resources. It has been designed in such a way so as to allow vocabularies to be layered on top of it, e.g. RDFS (RDF Schema). Being an abstract data model, RDF needs a concrete syntax so that it can be used and processed by software programs. Furthermore, it is much easier to represent information in RDF in the abstract form of node and directed arc graphs, where nodes are either resources or values, and directed arcs are properties.

Every arc in an RDF abstract data model represents a *statement*. A statement asserts a fact about a resource and has three parts: a *subject-predicate-object* triple. The subject is the resource from which the arc leaves. The predicate is the property that labels the arc. The object is the resource or literal to which the arc points. An RDF abstract data model is a set of statements i.e. a set of subject-predicate-object triples. With RDFS, one can define the vocabulary, specify which properties apply to which kinds of objects and what values they can take, and describe the relationships between objects. RDFS brings to the fore the semantics of the information held in RDF. Not only that, RDFS makes the semantics of information held machine-processable.

In our project, the ontology modelling language of choice is OWL. This is currently becoming the de facto standard in ontology modelling languages, superseding its mother-language, DAML+OIL. OWL is built upon the RDF and RDFS and further adds to the vocabulary set of both RDF and RDF Schema. New vocabulary includes the notion of disjointness between classes, cardinality, equality, symmetry of properties, enumerated classes, amongst others. For a complete introduction to OWL, see [SWM04].

1.3 Role of ontology mapping in searching

Ontologies can be a big help in the area of searching. One must realise, though, that for each domain of discourse there will be many ontologies available. This is because of

the decentralised nature of the Web: there is no *one Ontology* to suit everyone's needs (within a domain); rather, various ontologies from different providers will emerge. Furthermore, if a user requests information across different domains, then a software artefact will have to deal with ontologies from these domains. Mechanisms are required in order to enable information sharing and mappings between ontologies this is commonly called the interoperability problem.

Broadly speaking there are three architectures relating ontologies to one another and to instance data [W+01]: a single ontology architecture, a multiple ontology architecture, and a hybrid architecture. It was clear from past research that only the hybird architecture is suitable for a large scale web based application. Utilising ontologies will allow for the idea of semantic searching for a concept rather than a keyword searching. For this, ontology mapping will have to be developed. Ontology mapping highlights semantic similarities between ontologies.

1.4 Contributions of the paper

In developing our Semantic Web based search engine, we have put together some practical solutions to the above three issues.

First, we use RDF graphs to represent abstract ontologies and ontology instances and then convert each graph into an ontology file using Jena (Jena API, a Semantic Web toolkit from Hewlett-Packard Labs, http://jena.sourceforge.net/). Second, we modified an ontoligy mapping program by Euzenat [Euz04], which produces a file containing mapping pairs from two input ontology files. Third, a search engine is developed which takes a set of keywords as inputs, then searches a set of files in the RDF format. The returned result contains not only the URIs of documents containing the keywords searched, but also the URIs of documents containing words matched through ontology alignment. Therefore, the search is more semantic driven.

The rest of the paper is organized as follows. In Section 2, we discuss how to use Jena to create individual ontologies from ontology graphs. In Section 3, we investigate ontology mapping approaches and present our method of mapping used in the project. In Section 4, we introduce our search engine based on the Semantic Web and discuss our preliminary experimental results, and in Section 5, we conclude the paper.

2 Ontology creation

2.1 Ontology Creation

As mentioned earlier, a simple way to view RDF is as a graph of directed arcs and nodes. Figure 1 below contains two graphs representing two sets of instance data built from two ontology structures named as *ontology3* and *ontology4* in the following experiments. Similar instance data have been generated and used throughout this project. In total, we created six different ontologies for personal information including *name, date of birth*, and *address*. The ontology instance on the left has the *address* as a property which is another resource and has the *forename, surname* as separate properties having *literal* values. The ontology instance on the right, however, takes the *address* as a property with

a literal value, and has a property *fullname* instead of separating it into two properties. Resources are shown as ellipses and identified by Uniform Resource Identifier (URI) and literal values are in rectangles. Also, in Figure 1, we have used the XML qname form [BPM04] which is how a URI is commonly referred to in RDF/XML serialization. The part before ":" is called the namespace and the part after ":" is a local name within the specified namespace.

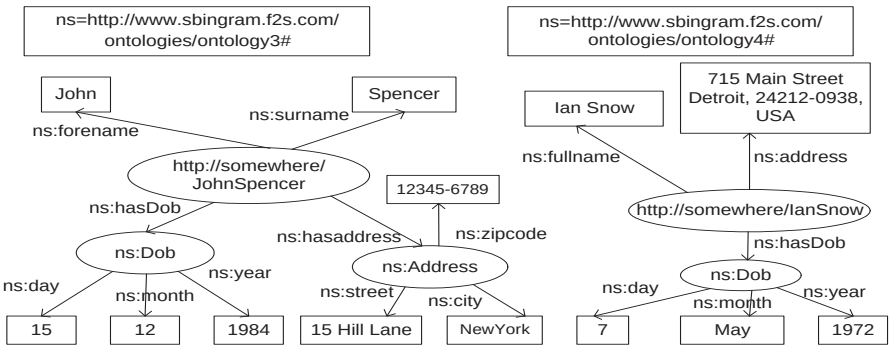

Fig. 1. Two ontology instances for personal information

Each RDF instance graph needs to be modelled in a proper format for retaining and searching the information in it. We store each of such graphs in a RDF file with suffix ".rdf". The instance data in Figure 1 right is therefore represented by the following file, with file name *ont4ian.rdf*, because the underlying ontology used is named *ontology4* in our experiment and the person to be modelled is called *Ian Snow*.

```
⟨?xml version="1.0"?⟩
⟨rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.sbingram.f2s.com/ontologies/ontology4#"⟩
  ⟨rdf:Description rdf:about="http://somewhere/IanSnow"⟩
    ⟨fullname⟩Ian Snow⟨/fullname⟩
    ⟨address⟩715 Main Street, Detroit, 24212-0938, USA⟨/address⟩
    ⟨hasDob⟩
      ⟨rdf:Description rdf:about="Dob"⟩
        ⟨day⟩7⟨/day⟩
        ⟨month⟩May⟨/month⟩
        ⟨year⟩1972⟨/year⟩
      ⟨/rdf:Description⟩
    ⟨/hasDob⟩
  ⟨/rdf:Description⟩
⟨/rdf:RDF⟩
```

2.2 Serialise Ontologies

An ontology can be represented as a graph similar to the ones in Figure 1, except that the values for literal properties are empty. Once an ontology graph in RDF is created,

it needs to be translated into a proper form of ontology. We use OWL as the ontology representation language, and deploy the methods in Jena API to first translate an ontology graph into a Java program which mirrors the graph and then run the program to create an OWL file with suffix *.owl*. The following is the ontology file in RDF/XML serialization form for the instance data on the right in Figure 1.

```
⟨ rdf:RDF
    xmlns="http://www.sbingram.f2s.com/ontologies/ontology4#"
    xml:base="http://www.sbingram.f2s.com/ontologies/ontology4#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"⟩
  ⟨ owl:Ontology rdf:about=""/⟩
  ⟨ owl:Class rdf:ID="Person"/⟩
  ⟨ owl:Class rdf:ID="Dob"/⟩
  ⟨ owl:ObjectProperty rdf:ID="hasDob"⟩
    ⟨rdfs:domain rdf:resource="#Person"/⟩
    ⟨ rdfs:range rdf:resource="#Dob"/⟩
  ⟨ /owl:ObjectProperty⟩
  ⟨ owl:DatatypeProperty rdf:ID="fullname"⟩
    ⟨ rdfs:domain rdf:resource="#Person"/⟩
    ⟨ rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/⟩
  ⟨ /owl:DatatypeProperty⟩
  ⟨ owl:DatatypeProperty rdf:ID="Address"⟩
    ⟨ rdfs:domain rdf:resource="#Person"/⟩
    ⟨ rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/⟩
  ⟨ /owl:DatatypeProperty ⟩
  ⟨ owl:DatatypeProperty rdf:ID="day"⟩
    ⟨ rdfs:domain rdf:resource="#Dob"/⟩
    ⟨ rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#positiveInteger"/ ⟩
    ⟨ /owl:DatatypeProperty⟩
    ⟨ owl:DatatypeProperty rdf:ID="month"⟩
    ⟨ rdfs:domain rdf:resource="#Dob"/⟩
    ⟨ rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/⟩
    ⟨ /owl:DatatypeProperty⟩
    ⟨ owl:DatatypeProperty rdf:ID="year"⟩
    ⟨ rdfs:domain rdf:resource="#Dob"/⟩
    ⟨ rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#positiveInteger"/⟩
  ⟨/owl:DatatypeProperty⟩
⟨ /rdf:RDF⟩
```

Once ontology models are created and filed, we will need to create some instance data to facilitate search. The instance data represent the information sources that are searched using our search engine. Eight information sources were created in our experiment: two based on ontology 1, two based on ontology 2, and one based on each of the remaining four ontologies. Each information source is a valid RDF file with suffix *.rdf* similar to the one below Figure 1.

3 Generating Ontology Mappings

Currently, there are many ontology mapping algorithms available but few have a concrete implementation. Some are an intrinsic part of the larger process of ontology merging, whereby two smaller ontologies are merged together to form one larger ontology, others exist as a standalone component that can be individually run to output a set of mappings.

Another major difference between ontology mapping methods is with regard to their level of automation. A zero level of automation indicates that the ontology mapping process is done by hand by a knowledge expert. Methods are referred to as semi-automatic if they require a certain amount of human involvement in the mapping process. Examples of semi-automatic ontology mapping methods are

To generate the axioms that map one ontology on to another, we use Euzenat's API for Ontology Alignment [Euz04]. Euzenat's program was run from the command line. The mapping algorithm used was *SubsDistNameAlignment* . The threshold for each mapping was maintained at 0.6 and the output was instructed to be in OWL format. By default, Euzenat's API outputs axioms in OWL format including statements (using $\langle$ owl:imports rdf:resource= "..."/$\rangle$) to import the two mapped ontologies. Should this axiom file be read in with the import statements included, Jena would create a model that was the union of all three documents (the two ontologies and the axiom file). In a real-world scenario this union model could be very large, therefore, to save memory on the machine on which the search engine is deployed, we have altered Euzenat's API so that the two import statements are omitted from the mapping output. The command that is executed from the command line in order to produce each axiom file is as follows:

```
PROMPT :> java  −jar lib/procalign.jar
−i fr.inrialpes.exmo.align.impl.method.SubsDistNameAlignment
file : /C : /pathTo/onto1.owl,  file : /C : /pathTo/onto2.owl  −t .6
−r fr.inrialpes.exmo.align.impl.renderer.OWLAxiomsRendererVisitor
−o outputSubdirectoryRelativeToCurrentDirectory/outputFilename.owl
```

In this way, all the mapping axioms are known. It should be pointed out that this method may only be feasible for a small size application where storing all the mapping axioms is practical. For a large size of application or for random information searching on the Semantic Web with a huge number of ontology collections, mapping all the pairs of ontologies may not be practical. Therefore, alternative mapping approaches that can generate mappings at run-time would be more appropriate.

4 The search Engine

4.1 System architecture and main functions

To make the search engine as user-friendly as possible, the user view was coded as JavaServer Pages (JSP). The heart of the system is the implementation of the semantic search engine. The search engine is in the form of a Java Servlet, which executes on the server after being invoked by a client request.

It is an important point to note that the search engine will not need to have access to the ontologies themselves, rather, due to the properties associated with the URI naming scheme, the search engine will simply require access to the axiom files that the mapping stage has generated. The overall architecture of the system is described below in Figure 2.

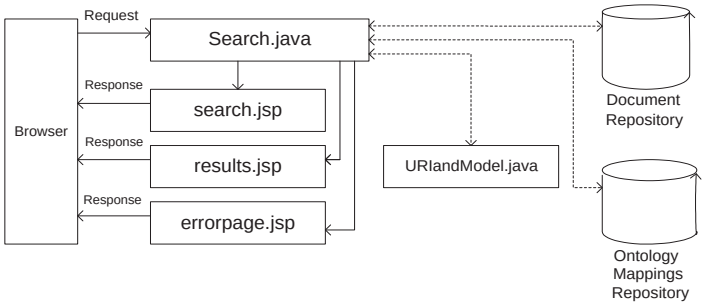

Fig. 2. The architecture of the search engine

The semantic search algorithm can be easily divided into three main steps with each succeeding step dependent on the results of the previous step(s).

(A) *Keyword Matching*. In this step of the algorithm, the available instance data is interrogated using a keyword-based matching approach which is similar to how current search engines work. If a match is found, the URI of the matching field is stored. This step creates two subsets of the set of available instance data: *a Matched set*, where keyword matches were found; and, *a Pending set*, where no keyword matches were found.

(B) *Find Relevant Mappings*. Based on the fact that the ontology mappings have already been generated, the mappings in the axioms files are searched for instances of each URI stored in the *Matched set* from (A). Due to the properties intrinsic to the URI naming system, it is guaranteed that all references in any mapping file to a URI contained in the *Matched set* will be relevant to either the *subject*, *predicate* or *object* at hand. This step creates a subset of the set of mappings.

(C) *Interrogate Pending Models with Relevant Mappings*. In this final step each *subject-predicate-object statement* in each model in the *Pending set* is compared with each relevant mapping deduced in Step (B). Upon comparison, if a match is found then the model in the *Pending set* to which the current statement belongs is added to the *Matching set* and the step moves on the next model in the *Pending set*. If the set of relevant mappings is exhausted with no match found in the current model of the *Pending set*, then the current model in the *Pending set* is not included in the results set.

In addition to the semantic search, we also included the option of an ordinary only keyword-based search. This functionality is to allow the user to compare the results set returned from an ordinary keyword-based search with the results set returned from a semantic search. Some users may not want the semantic search algorithm to execute; instead wanting exact keyword matches. In this case, enabling this option will increase the run-time performance of the search engine. The information sources whose URIs will form the results set will, however, be in the Semantic Web format.

4.2 Experimental results

To get a measure of the effectiveness of the semantic search algorithm, we now examine the contents of each of the created sets after each step in the algorithm. An effective semantic search is not just one that returns the correct set of results; the semantic matching must also occur on correct subject, predicates or objects. we will also compare the set of results returned from a keyword-based search to that from a semantic search.

To maintain consistency, the four experiments were carried out in precisely the same manner. Each experiment has been given an id, *Search 1* through to *Search 4* , and each search has a corresponding keyword that was entered by the user. These keywords are *Address*, *postcode*, *Mark,* and *phone number* respectively. There are eight instance data files generated from six different ontology models for the search engine to search for information.

Steps (A), (B), and (C) correspond to the different parts of the semantic search algorithm detailed above. Statements, i.e., subject-predicate-object triples, are enclosed in square brackets like [subjectURI, predicateURI, objectURI].

Search 1: Address After step (A), *Matched Statements* (fields in bold are the field matching occurred on):

[http://somewhere/MarkIngram,
http://www.sbingram.f2s.com/ontologies/ontology1#hasAddress, Address]
[**Address** http://www.sbingram.f2s.com/ontologies/ontology1#country, “UK”]
[**Address**, http://www.sbingram.f2s.com/ontologies/ontology2#street1, “18 Knockmoyle”]
[**Address**, http://www.sbingram.f2s.com/ontologies/ontology2#street1, “18 Knockmoyle”]
[**Address**, http://www.sbingram.f2s.com/ontologies/ontology3#zipcode, “12345-6789”]
[http://somewhere/IanSnow,
http://www.sbingram.f2s.com/ontologies/ontology4#address , “715 Main Street, Detroit, 24212-0938, USA”]
[**Address**, http://www.sbingram.f2s.com/ontologies/ontology5#city, “Oslo”]
[**Address**, http://www.sbingram.f2s.com/ontologies/ontology6#country, “USA”]

Pending: none
After step (B) *Mappings: none*
After step (C) *Statements added to Matched: none*
URIs returned: Keyword-based search (left): Semantic search (right) :

C:\repository\ont1mark.rdf	file:\C:\repository\ont1mark.rdf
C:\repository\ont1sam.rdf	file:\C:\repository\ont1sam.rdf
C:\repository\ont2jenna.rdf	file:\C:\repository\ont2jenna.rdf
C:\repository\ont2nicola.rdf	file:\C:\repository\ont2nicola.rdf
C:\repository\ont3john.rdf	file:\C:\repository\ont3john.rdf
C:\repository\ont4ian.rdf	file:\C:\repository\ont4ian.rdf
C:\repository\ont5jack.rdf	file:\C:\repository\ont5jack.rdf
C:\repository\ont6suzanne.rdf	file:\C:\repository\ont6suzanne.rdf

Search 2: postcode After step (A), *Matched Statements* (fields in bold are the field matching occurred on):

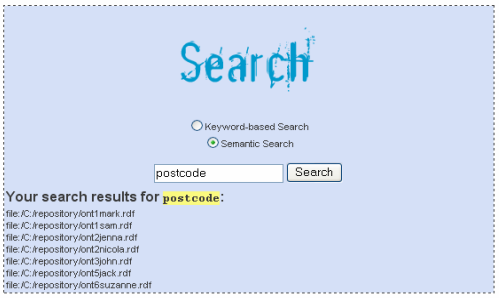

Fig. 3. The screen shot of searching result of *postcode*

[Address, **http://www.sbingram.f2s.com/ontologies/ontology1#postcode** , "BT41 1HT"]
[Address, **http://www.sbingram.f2s.com/ontologies/ontology1#postcode**, "BT41 1HT"]
[Address, **http://www.sbingram.f2s.com/ontologies/ontology2#postcode**, "BT41 4HE"]
[Address, **http://www.sbingram.f2s.com/ontologies/ontology2#postcode**, "BT41 4HE"]

Pending:

file:\C:\repository\ont3john.rdf
file:\C:\repository\ont4ian.rdf
file:\C:\repository\ont5jack.rdf
file:\C:\repository\ont6suzanne.rdf

After step (B) *Mappings:*

1. [http://www.sbingram.f2s.com/ontologies/ontology1#postcode,
2. http://www.w3.org/2002/07/owl#equivalentProperty,
3. http://www.sbingram.f2s.com/ontologies/ontology2#postcode]
4. [http://www.sbingram.f2s.com/ontologies/ontology1#postcode,
5. http://www.w3.org/1999/02/22-rdf-syntax-ns#type,
6. http://www.w3.org/2002/07/owl#DatatypeProperty]
7. [http://www.sbingram.f2s.com/ontologies/ontology1#postcode,
8. http://www.w3.org/2002/07/owl#equivalentProperty,
9. http://www.sbingram.f2s.com/ontologies/ontology3#zipcode]
10. [http://www.sbingram.f2s.com/ontologies/ontology2#postcode,
11. http://www.w3.org/2002/07/owl#equivalentProperty,
12. http://www.sbingram.f2s.com/ontologies/ontology1#postcode]
13. [http://www.sbingram.f2s.com/ontologies/ontology3#zipcode,
14. http://www.w3.org/2002/07/owl#equivalentProperty,
15. http://www.sbingram.f2s.com/ontologies/ontology1#postcode]

: (many more lines are deleted which give out all the mapping pairs)

After step (C) *Statements added to Matched:*

[Address, http://www.sbingram.f2s.com/ontologies/ontology3#zipcode, "12345-6789"]
[Address, http://www.sbingram.f2s.com/ontologies/ontology5#areacode, "793B0NN"]
[Address, http://www.sbingram.f2s.com/ontologies/ontology6#zipcode, "09893-2812"]

URIs Returned: Keyword-based search (left), Semantic search (right)

C:\repository\ont1mark.rdf	file:\C:\repository\ont1mark.rdf
C:\repository\ont1sam.rdf	file:\C:\repository\ont1sam.rdf
C:\repository\ont2jenna.rdf	file:\C:\repository\ont2jenna.rdf
C:\repository\ont2nicola.rdf	file:\C:\repository\ont2nicola.rdf
	file\C:\repository\ont3john.rdf
	file\C:\repository\ont5jack.rdf
	file:\C:\repository\ont6suzanne.rdf

Search 3: Mark After step (A) *Matched Statements:* (fields in bold are the field matching occurred on)

[**http://somewhere/MarkIngram**, http://www.sbingram.f2s.com/ontologies/ontology1#hasDob, Dob]

Pending:

file:/C:/repository/ont1sam.rdf	file:/C:/repository/ont2jenna.rdf
file:/C:/repository/ont2nicola.rdf	file:/C:/repository/ont3john.rdf
file:/C:/repository/ont4ian.rdf	file:/C:/repository/ont5jack.rdf
file:/C:/repository/ont6suzanne.rdf	

After step (B) *Mappings: none*
After step C *Statements added to Matched: none*
URIs Returned: Keyword-based search (left), Semantic search (right)

C:\repository\ont1mark.rdf	file:\C:\repository\ont1mark.rdf

Search 4: phone number *After step (A): Matched statement: none*
Pending:

file:\C:\repository\ont1mark.rdf	file:\C:\repository\ont1sam.rdf
file:\C:\repository\ont2jenna.rdf	file:\C:\repository\ont2nicola.rdf
file:\C:\repository\ont3john.rdf	file:\C:\repository\ont4ian.rdf
file:\C:\repository\ont5jack.rdf	file:\C:\repository\ont6suzanne.rdf

After step (B) *Mappings: none*
After step (C) *Statements added to Matched: none*
URIs Returned: Keyword-based search: *none*; Semantic search: *none*

4.3 Discussion of Results

The searches carried out were chosen specifically to show the search algorithm working in four discrete scenarios: **Search 1** shows the situation when every available information source has a keyword matching to the user's query. **Search 2** shows the situation when some of the available information sources contain a matched keyword and the results set has to be augmented by inspection of ontology mappings. **Search 3** shows the situation if only one of the available information sources has a keyword match and, even after inspection of the ontology mappings, no other matches can be found. **Search**

4 shows the situation when the user's query does not match anything in any of the available information sources.

In **Search 1**, once we found one matching axiom between two ontologies, we stop finding other matchings, since one matching axiom is enough to decide that the related instance data should be included in the set of results. It should be pointed out that the matched axioms returned in this search changes from ontology to ontology. This is due to the practical implementation of Jena. Jena stores statements as triples in a hash table. It does not preserve order. Model.listStatements() in Jena iterates over a hash table and returns things in whatever order they are in the hash table. Blank nodes get internal identifiers - this is one cause of hash order variation. It is not even guaranteed the same order reading one file across different runs of our application or within two readings of a file in the same application run. Nevertheless, from the point of view of search, the end result is the same since we only need to find relevant URIs.

For each experiment it can be seen that for a keyword-based search the set of results returned to the user contains exactly the same number of URIs as there were keyword matches after step (A) of the semantic search algorithm. This is precisely what is to be expected because step (A) of the algorithm is simply a naive keyword search of the available information sources. Furthermore, it can be seen that the results set for a semantic search contains the URIs expected from the keyword search as well as the URI of any information source that had a matching statement listed in step (C) of the search algorithm. Therefore, the result set of the semantic search is the union of the keyword-based search results set and the results set created after inspection of the available ontologies.

$\{\mathsf{SemanticSearchRS}\} = \{\mathsf{KeywordSearchRS}\} \cup \{\mathsf{OntologySearchRS}\}$

This is precisely what a semantic search is aimed to achieve: not only will a user's search return URIs based on keyword matches but also returned will be URIs based on concepts that are deemed similar to the keyword matches by ontology mappings.

5 Conclusions

We have come across two semantic search engines during the project. One is based on [Euz04] (http://align.deri.org:8080/deri/align.jsp). Unfortunately, this search engine has been down for some time, and we were unable to get an access to it in the last few months. The other semantic search engine can be found at http://pear.cs.umbc.edu/swoogle/. Swoogle does not carry out any ontology mapping; instead it is a keyword-based search engine tailored specifically for documents encoded in the Semantic Web format. Like Google, Swoogle crawls the web for Semantic Web documents and indexes them. Swoogle provides a way for Semantic Web developers and researchers to check to see whether a concept has already been defined or not. As stated by the author's of Swoogle: "... today's search engines deal with SWDs [Semantic Web Documents] poorly, if at all, since they have been developed to process text documents. Most make no attempt to parse XML documents into appropriate tokens and none take advantage of the structural and semantic information encoded in a SWD. A search engine customized for SWDs, especially for SWDs that define or extend ontologies, might be very useful for both semantic web developers as well as software agents, tools and services."

In this paper, we reported a preliminary implementation of a semantic search engine which utilizes a number of independent software packages (e.g., Jena, and Euzenat's ontology mapping). The search engine returned correct results sets within the restricted domain of discourse of personal information but dealt mainly with equivalence relations between concepts.

In this project, the information sources used were very small and the investigation only dealt with similarities in the strings and/or substrings of subjects, predicates and objects. Information sources that included subclass and super class relationships, cardinality variations and enumerations would verify the robustness of the search engine in our future development.

To make the search engine more scalable, the generation of ontology mappings phase would need to be investigated more closely. If the search engine is to be used for a small medical research lab where the information sources grew slowly and the ontology repository even more slowly, then the generation of ontology mappings phase could remain as it is. However, if this semantic search engine is for general release on the Internet, then a much more automatic ontology mapping generation process is required. Mapping on the semantic similarities of conepts in different natural languages can also be incorporated. This will be another focus of our next step of research.

References

[BPM04] P Biron, K Permanente, and A Malhotra. XML Schema Part 2: Datatypes Second Edition. http://www.w3.org/TR/xmlschema-2/#QName, 2004.

[Euz04] J Euzenat. An API for ontology alignment. http://co4.inrialpes.fr/align/align.pdf, 2004.

[ESt04] M Ehrig and S Staab. Quick ontology mapping (QOM). *GI Jahrestagung* (1):356-361, 2004.

[ESu04] M Ehrig and Y Sure, Ontology mapping: An integrated approach. http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/2004_mapping_TR.pdf.

[Gru93] T Gruber. A translation approach to portable ontology specifications. *knowledge Acquisition*, 5:199-200, 1993.

[Li03] J Li. A lexicon-based ontology mapping tool. *Conference of Information Interpretation and Integration*, 2003.

[MMM04] F Manola, E Miller, and B McBride. RDF Primer. http://www.w3.org/TR/rdf-primer.

[NM03] N Noy and M Musen. The PROMPT Suite: interactive tools for ontology merging and mapping. ttp://www-smi.stanford.edu/pubs/SMI_Reports/SMI-2003-0973.pdf, 2003.

[SWM04] M Smith, C Welty and D McGuinness OWL Web ontology language guide. http://www.w3.org/TR/owl-guide/.

[Stu03] G Stumme et al. The Karlsruhe view on ontologies. Technical Report, University of Karlsruhe, Institute AIFB, 2003.

[Su02] X Su. A text categorisation perspective for ontology mapping. http://www.idi.ntnu.no/ xiaomeng/paper/Position.pdf, 2002.

[Su03] X Su. Ontology mapping through analysis of model extension. *Proc. of CAiSE'03*.

[W+01] H Wache, T Voegele, U Visser, H Stuckenschmidt, G Schuster, H Neumann, and S Huebner. Ontology-based integration of information - A survey of existing approaches. *IJCAI'01 Workshop on Ontologies and Information Sharing*, 108-117. 2001.