
MSc in Machine Learning and Data Mining Project

Interim Project Report

**A Search Engine Based on the
Semantic Web**

Peng Wang

CS Supervisor: Peter Flach

External Supervisor: Simon Price

May, 2003

TABLE OF CONTENTS

1. Introduction	3
2. Background	4
2.1 Introduction to URI	4
2.2 Introduction to HTTP	4
2.3 XML	5
2.4 RDF	8
2.5 DAML	12
2.6 Ontology Matching	15
2.7 Similarity Measures	19
2.8 Toolkits	22
3. Project Plan	
3.1 Aims and Objectives	23
3.2. Initial Design	23
3.3 Project Schedule	26
4. Bibliography	
5. References	

Part I

1. Introduction

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

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

Nowadays, there are a huge amount of resources on the Web, which raises a serious problem of accurate search. This is because data in HTML files is useful in some contexts but meaningless under other conditions. In addition, HTML cannot provide description of data encapsulated in it. For example, we want to find an address’ details and know its postcode. Since the names of the postcode system are different in many countries and the Web doesn’t represent this relationship, we may not get what we expect. By contrast, in the Semantic Web, we can indicate this kind of relationship such as zip code is equivalent to postcode. So when the majority of data on the Web are presented in this form, it is difficult to use such data on a large scale [3]. Another shortcoming is that today’s Web lacks an efficient mechanism to share the data when applications are developed independently. Hence, it is necessary to extend the Web to make data machine-understandable and integrated and reusable across various applications [1].

To make the Semantic Web work, well-structured data and rules are necessary for agents to roam the Web [2]. XML and RDF are two important technologies: we can create our own structures by XML without indicating what they mean; RDF uses sets of triples which express basic concepts [2, 5]. DAML is the extension of XML and RDF.

The aim of this project is to develop a search engine based on ontology matching within the Semantic Web. It uses the data in Semantic Web form such as DAML or RDF. When the user input a query, the program accepts the query and transfers it to a machine learning agent. Then the agent measures the similarity between different ontologies, and feedback the matched item to the user.

The following sections are organized into 4 parts: Section 2 gives the background knowledge

of some relative techniques such as HTTP, XML, RDF, DAML and ontology. The project plan is given in Section 3, including the aims, the objectives, the initial design and project schedule. Section 4 and Section 5 are bibliographies and references respectively.

2. Background Knowledge

2.1 Introduction to URI - Uniform Resource Identifier

URIs are strings that can identify resources in the Web [5]. By using URIs, we can use the same simply naming way to refer to resources under different protocols: HTTP, FTP, GOPHER, EMAIL etc [5]. URLs (Uniform Resource Locator), a widely used type of URIs, are very commonly used in the web, which are addresses of resources. Although often referred to as URLs, URIs can also refer to concepts in the Semantic Web [9], e.g. suppose you have a book with the title “Machine Learning”, then its URI will look like this: <http://www.cs.bris.ac.uk/home/pw2538/book/title#machinelearning>

Here are some other examples of URIs:

uuid:04b749bf-3bb2-4dba-934c-c92c56b709df is a UUID, which stands for a Universal Unique Identifier. UUID can be got by combining the time and the address of your Ethernet card or a random number, which is then a unique identifier [9].

<mailto:pw2538@bristol.ac.uk> identifies the mail address of a person.

Note that everything on the Web has a unique URI.

2.2 Introduction to HTTP

The Hypertext Transfer Protocol (HTTP) is a generic and stateless protocol for distributed, collaborative, hypermedia information systems [13]. It allows performing operations on resources which are identified by URIs. It has been widely used for more than one decade and now comes to version 1.1. It has four main methods:

- a. **Get** means get the information that is identified by the request URI [14]. Usually it is the action we take when browsing sites and clicking hyperlinks -- we ask the web server for the resources that is identified by Request-URLs.
- b. **Post** means make a request to the web server so that the server accepts the resources encapsulated in the request, which will be the new subordinate of the resource identified by the Request-URI in the Request-Line [14]. Usually it is what we do when filling and sending the HTML form to the server to buy something like CDs, books etc. -- making a

request of the server.

- c. **Put** means make a request that sends updated information about a resource if the resource identified by the Request-URI exists, otherwise the URI will be regarded as a new resource [14]. The main difference between the POST and PUT requests lies in the different meaning of the Request-URI. In a POST request, the URI is to identify the resource that will handle the enclosed entity. As for the PUT request, the user agent knows what URI is its aim and the web server cannot redirect the request to other resources. Unfortunately most web browsers don't implement this functionality, which makes the Web, to some extent, a one-way medium [9].
- d. **Head** is similar to GET except that the server don't return a message-body in the response. The benefit of this method is that we can get meta-information about the entity implied by the request without transferring the entity-body itself. We can use this method to check if hypertext links are valid, or if the content is modified recently [14].

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

2.3 XML

Extensible Markup Language (XML) is a subset of SGML (the Standard Generalized Markup Language) [12], i.e. it is totally compatible with SGML. But it is simple and flexible. It's original aim to tackle the problems of large-scale electronic publishing. However, it is also very important in data exchange on the Web. Despite its name, XML is not a markup language but a set of rules to build markup languages [17].

2.3.1 Markup language

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

2.3.2 Why XML

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

complicate to be implemented in web browsers although it can do everything XML can do.

2.3.3 XML Documents

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

```
<?xml version='1.0'?>
<!-- This file represents a fragment of a book store inventory database -->
<bookstore>
  <book genre="science">
    <title>Machine Learning</title>
    <author>
      <first-name>Tim</first-name>
      <last-name>Mitchell</last-name>
    </author>
    <price>28.99</price>
  </book>
  <book genre="travel">
    <title> Family Fun Vacation Guides </title>
    <author>
      <first-name>Jill</first-name>
      <last-name>Mross</last-name>
    </author>
    <price>12.57</price>
  </book>
  <book genre="philosophy">
    <title>The Gorgias</title>
    <author>
      <name>Plato</name>
    </author>
    <price>9.99</price>
  </book>
</bookstore>
```

Figure 2.3.3.1: An example of XML document

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

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

2.3.4 Namespaces

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

ns-prefix:local-name

2.3.5 XML Schemas

XML itself does not do anything, i.e., it is just structure and store information. But if we need a program to process the XML document, there must be some constraints on sequence of tags, nesting of tags, required elements and attributes, data types for elements and attributes, default and fix values for elements and attributes and so on. XML Schema is an XML based alternative to Document Type Definition (DTD) [19]. There are some features of XML Schemas that outweigh DTD:

- a. XML Schemas support data types, which brings a lot of benefits, e.g. easy to validate the correctness of data, easy to work with databases, easy to convert data between different types.
- b. XML Schemas have the same syntax as XML so that it can benefit all features of XML.
- c. XML Schemas secure data communication since it can describe the data in a machine-understandable way.
- d. XML Schemas are extensible because they are actually XML and then share this feature of XML.
- e. Well-formed is not enough since it also may contain some semantic confusion which can be caught by XML Schemas.

2.3.6 Well-Formed and Valid Documents

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

- a. There is one, and only one, root element [18].
- b. Each tag must be closed [18].
- c. "Tag names are case-sensitive" [18].

A well-formed XML document is valid only if it refers to a proper DTD or XML Schema so that the document obeys the constraints of that DTD or XML Schema. [18].

2.4 RDF

2.4.1 Metadata

Metadata is information about information [20], which is widely used in real-world for searching. For example, you want to borrow some books on computer from a library. Usually a library will provide a lookup system which allows you to list books by author, title, subject and so on. This list contains lots of useful information: author, title, ISBN, date and most important, location of the book. You need some information (the book's location) you want to know and you use metadata (information about information, in this case: author, title and subject) to get it. However, metadata is not necessary [20]: you can lookup the book you want to find one by one among all books in the library. Obviously this is not a wise way. In addition, the use of metadata is not just for searching although searching is the most common aim of metadata. There is some other useful information behind the scenes, which are important to business.

2.4.2 What is RDF?

Resource Description Framework is a framework for processing metadata [20] and it describes relationships among resources with properties and values [23]. It is built on the following rules:

- a. **Resource:** Everything described by RDF expressions is called a resource [22]. Every resource has a URI and it may be an entire web page or a part of a web page [20, 22]
- b. **Property:** “A property is a specific aspect, characteristic, attribute, or relation used to describe a resource” – W3C, Resource Description Framework (RDF) Model and Syntax Specification [22]. Note that a property is also a resource since it can have its own properties.
- c. **Statements:** A statement combines a resource, a property and a value [22]. These three individual parts are known as the “subject”, “predicate” and “object” [20]. For example, “The Author of <http://www.cs.bris.ac.uk/home/pw2538/index.html> is Peng Wang” is a statement. Note that value can be either a string or another resource [22].

2.4.3 Examples

Statements can be represented as a graph in RDF.

First consider a simple example:

Peng Wang is the author of the resource <http://www.cs.bris.ac.uk/home/pw2538/index.html>

This sentence has the following parts:

Subject (Resource)	http://www.cs.bris.ac.uk/home/pw2538/index.html
Predicate (Property)	Author
Object (literal)	“Peng Wang”

Figure 2.4.3.1: Divide the sentence into 3 parts

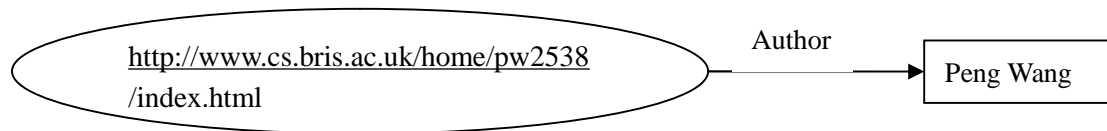


Figure 2.4.3.2: Simple node and arc diagram

The direction of the arrow is always from the subject to the object of the statement. And the graph can be read in the way: "<subject> HAS <predicate> <object>" [22], i.e. "<http://www.cs.bris.ac.uk/home/pw2538/index.html> has the author Peng Wang".

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

<http://www.cs.bris.ac.uk/home/pw2538/terms/author>

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

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

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

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

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

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

Prefix *pwterms*., namespace URI: <http://www.cs.bris.ac.uk/home/pw2538/terms>

```

1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.           xmlns:pwterms=" http://www.cs.bris.ac.uk/home/pw2538/terms">
4.   <rdf:Description rdf:about="http://www.cs.bris.ac.uk/home/pw2538/index.html">
5.     <pwterms:author>Peng Wang</pwterms:author>
6.   </rdf:Description>
7. </rdf:RDF>

```

Figure 2.4.3.3: RDF for a Simple RDF Statement

Now consider a more complicate example:

The individual referred to by student id pw2538 is named Peng Wang and has the email address pw2538@bristol.ac.uk. This individual is the author of the resource <http://www.cs.bris.ac.uk/home/pw2538/index.html>.

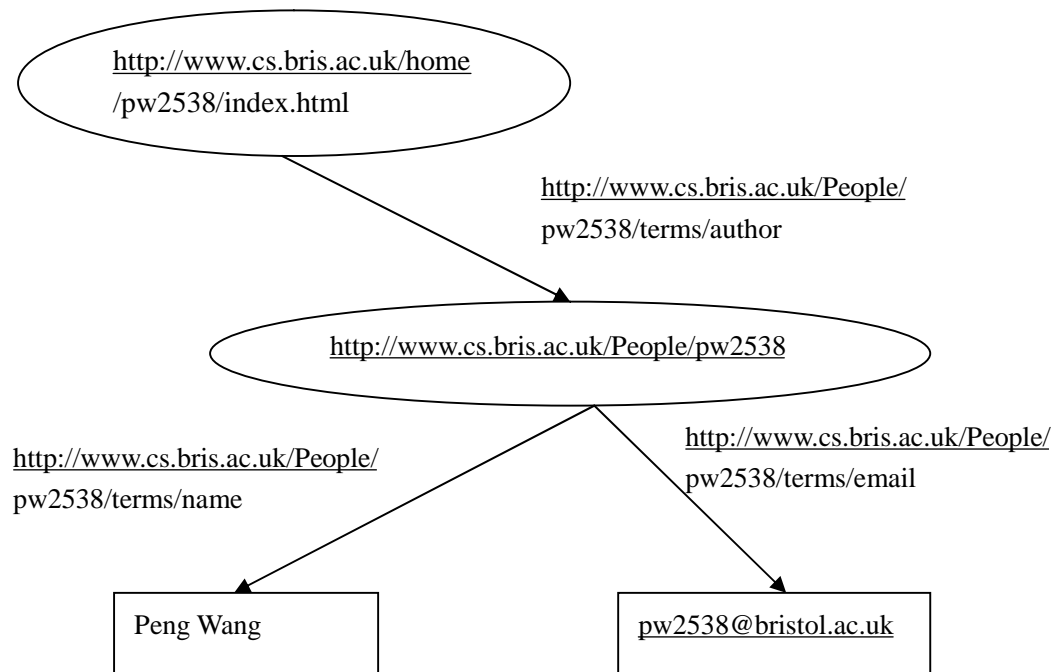


Figure 2.4.3.4: Structured value with identifier

```
1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.     xmlns:pwterms="http://www.cs.bris.ac.uk/home/pw2538/terms">
4.   <rdf:Description about="http://www.cs.bris.ac.uk/home/pw2538/index.html">
5.     <pwterms:author rdf:resource="http://www.cs.bris.ac.uk/People/pw2538"/>
6.   </rdf:Description>
7.   <rdf:Description about="http://www.cs.bris.ac.uk/People/pw2538">
8.     <pwterms:Name>Peng Wang</pwterms:Name>
9.     <pwterms:Email>pw2538@bristol.ac.uk</pwterms:Email>
10.  </rdf:Description>
11. </rdf:RDF>
```

Figure 2.4.3.5: RDF for a complicate RDF Statement

2.4.4 Why Not Just Use XML?

Since RDF is based on XML and XML can also represent the statements in a natural way, why not just use XML instead of using a new language RDF [20]? However, XML has some shortcoming when dealing with metadata:

- a. In XML documents, the order of elements is often very important and meaningful [20]. However, in metadata, this is redundant; for instance, we don't care whether a book is listed first when we look up in the library [20]. Furthermore, it will reduce the performance and efficiency if maintaining the correct order of data items [20].
- b. XML allows mixed structures like

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

Figure 2.4.4.1: Partial XML document with mixture structure

So data structures in XML will include the mixture of trees, graphs, and character strings [20]. In general, it requires more computation when dealing with these complicate structures. By contrast, RDF is more straightforward.

2.4.5 RDF Schemas

Although RDF can easily describe resources, we still need a mechanism to figure out what a specific term means and how it should be used [3, 10]. This is the function of the RDF vocabulary description language, RDF Schema. RDF Schema is a simple data-typing model for RDF [3] so that we can describe groups of related resources and the relationships among these resources [10, 23]. For example, we can say “pupil” is a type of “student” and “student” is a subclass of “people”.

Resources can be divided into “classes” which is composed of instances [23]. A class itself is also a resource which is usually identified by RDF URI References and can be described by RDF properties [23]. We often use the prefix “rdfs:” to indicate the term is RDF Schema term. “rdfs:Resource” is the root class of everything in RDF Schema. “rdf:type” is an instance of rdf:Property (class of RDF properties), and it means that a resource is an instance of a class. The property rdfs:subClassOf is an instance of rdf:Property that is used to state a class is a subclass of the other. **Figure 2.4.5.1** shows that “Animal” is the super-class in this RDF document, i.e. “Dog”, “Cat”, “Donkey” and “PersianCat” are all subclasses of “Animal”. Although “PersianCat” class doesn't directly indicate this relation, the relationship can be got from: “PersianCat” is the subclass of “Cat” and “Cat” is the subclass of “Animal”, then “PersianCat” is also the subclass of “Animal”. This is similar to the feature inheritance in the

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

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

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

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

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

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

</rdf:RDF>
```

Figure 2.4.5.1: The Animal Class Hierarchy in RDF/XML

2.5 DAML

The DARPA Agent Markup Language (DAML) Program started in 2000. DAML combines many language components of the Ontology Inference Layer (OIL) soon after it was started. The result of these efforts is DAML+OIL, a more robust language for general knowledge representation than RDF and RDFS. DAML is not a W3C standard, but many people in W3C participated in this program. DAML is kind of extension of RDF and RDFS, but it is not a

data model. It not only provides stronger abilities to express constraints in schemas but also can build general knowledge representation, i.e. it is also an ontology language.

2.5.1 Introduction to DAML

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

- a. **Properties:** DAML add a primitive “*DatatypeProperty*” that allows strict data types that defined in XML Schemas or user-defined data types e.g. float number, integer and so on. In DAML, a property can have multiple ranges, which brings rich flexibilities. Furthermore, DAML allows we declare a unique property, i.e. there are no two instances with same value. This is the function of a primitive “*daml:UniqueProperty*”. We can also describe the relation between two properties that are equivalent by either “*daml:samePropertyAs*” or “*daml:equivalentTo*”. In addition, there are more powerful features in properties in DAML, with which we can express relations such as “inverse”, “transitivity”. If A is the employer of B, then B is the employee of A. The properties “employer” and “employee” are the inverse of each other. This relation can be expressed with “*daml:inverseOf*”. The “transitivity” means that if A is a subset of B, and B is a subset of C, then A must be a subset of C. The property “*daml:TransitiveProperty*” is used to express this relation. More interestingly, DAML provides “*daml:onProperty*”, “*daml:hasValue*”, “*daml:hasClass*” and “*daml:toClass*” to restrict classes to a set of resources based on particular properties. Then we can make the rules for a specific class so that a resource can be a member of the class if and only if its properties must satisfy the requirements. “*daml:onProperty*” identifies the property to be checked. We can define property restrictions by its value with “*daml:hasValue*”, i.e. the property must have a particular value. “*daml:hasClass*” can be used to define property restrictions by the class of the values of a property, instead of its value. By contrast, “*daml:toClass*” is more restrictive since it requires that **all** the property values for a resource must be a particular class. However, a resource without property given “*daml:onProperty*” can also satisfy the condition. So this feature must be used very carefully.
- b. **Classes:** “*daml:Class*” is a subclass of “*rdfs:Class*” and DAML adds many wonderful features in it. We can build more expressiveness description of resources with these features. We can define an enumeration that cannot be implemented in RDF. In DAML, “*daml:oneOf*” element defines an enumeration. We can also define a closed list by declaring “*daml:oneOf*” to be the “*daml:collection*” parse type. Additionally, we can build some relations such as “disjoint”, “union” and “intersection”. Both “*daml:disjointWith*” and “*daml:disjointUnionOf*” can be used to assert there are no instances in common among classes. Non-exclusive boolean combinations of classes can be expressed with “*daml:unionOf*”. The “*daml:intersectionOf*” property can express intersection of the sets.

2.5.2 Why DAML

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

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

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

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

Figure 2.5.2.1: Book Example with RDF and RDFS

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

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

Figure 2.5.2.2: Book Example with DAML

2.6 Ontology Matching

2.6.1 Ontology

The word “ontology” is borrowed from philosophy. Its original meaning is “the branch of metaphysics that deals with the nature of being” -- The American Heritage[®] Dictionary of the English Language: Fourth Edition (2000). In AI domain, T. R. Gruber defined the term as a specification of a conceptualization [29], i.e. it is a description of concepts and relationships with a set of representational vocabulary. The aim of building ontologies is to share and reuse knowledge.

2.6.2 Ontology Matching

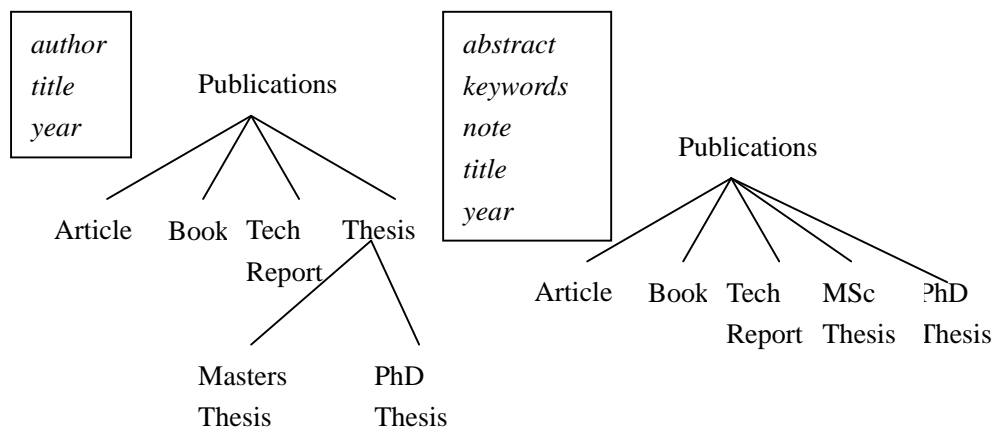


Figure 2.6.2.1 Two Publication Ontologies

Since the Semantic Web is built distributively, there are many different ontologies that describe the semantically equivalent things. Therefore it is necessary to map among elements of these ontologies if we want to process information in the Web scale. An ontology can be represented in taxonomy tree form where each node represents a concept with its attributes [30]. **Figure 2.6.2.1** shows two different publication ontologies. For example, the concept publication on the left of **Figure 2.6.2.1** has three attributes: author, title and year. The aim of ontology matching is to map the semantically equivalent elements. For example, “MastersThesis” maps to “MScThesis” in **Figure 2.6.2.1**. This is a one-to-one mapping [30], the simplest type. We can also map the different types of elements, e.g. a particular relation maps to a particular attribute. Mapping can be more complex if we want to map the combination of some elements to a specific element. For example, “FullName” maps to the combination of “FirstName” and “LastName”.

2.6.3 Approaches

Xiaomeng, in her position paper [33], provided an approach based on text categorization for ontology mapping. We should compare each element of an ontology with each element of the other ontology, and then determine a similarity metric per pair. Matched items are those whose similarity values are greater than a certain threshold.

Since mapping assertion is the core output of the mapping process, a meta-model for mapping is defined as **Figure 2.6.3.1** shows. This meta-model means that “a mapping assertion is an objectification of the relationship between two ontology elements and supports further description of that relationship” [33]. It has a mapping type, a mapping degree, an assertion source. Mapping degree is to rank the outputs, and mapping type is to indicate the relationship between two ontology elements. The assertion source gives the reason why the particular assertion is chosen.

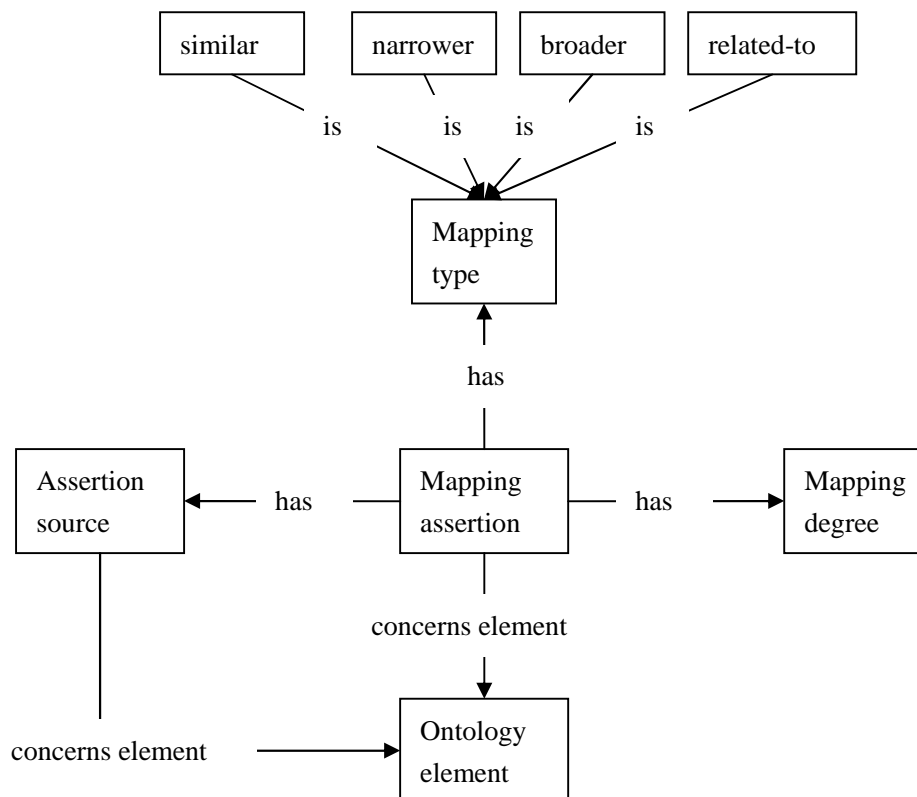


Figure 2.6.3.1 Mapping Assertion Meta-model

(Modified from “A Text Categorization Perspective for Ontology Mapping” [33])

As the **Figure 2.6.2.1** shows, an ontology can be regarded as a taxonomy tree of a domain and each node in the taxonomy can be regarded as a category which has documents assigned to it.

Suppose there are two ontologies O_1 and O_2 . A similarity measure $sim(a_i, b_j)$ is computed

for every node in A, where a_i belongs to A and b_j belongs to B. The node with the highest similarity will be ranked on top. $sim(a_i, b_j)$ will be computed with some information retrieval techniques.

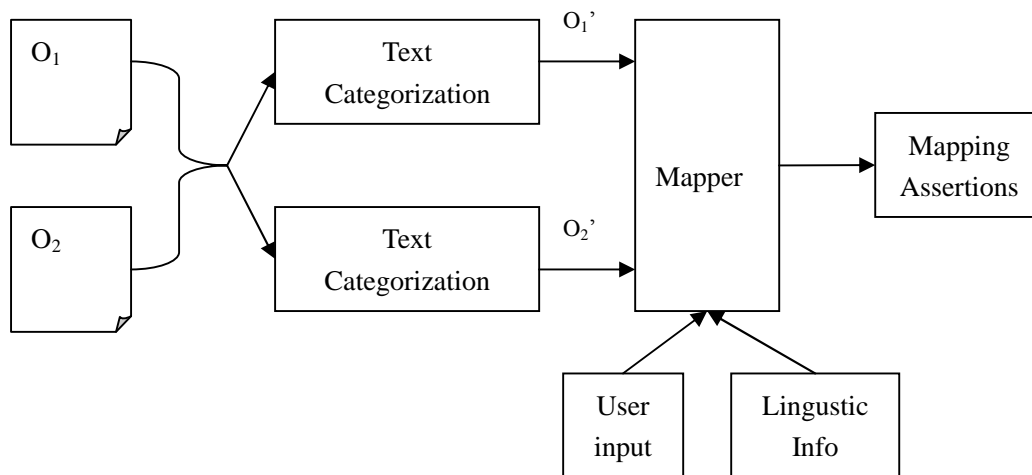


Figure 2.6.3.2: Architecture of Ontology mapping in Xiaomeng's perspective (Modified from "A Text Categorization Perspective for Ontology Mapping" [33])

Figure 2.6.3.2 shows the workflow of this approach. First we use some text categorization techniques (Naïve Bayes, Nearest Neighbour, for instance) to classify some documents to concept nodes of the ontology. Then we use the two temporary ontologies as the input and use some information retrieval techniques to produce the output.

In a paper published in WWW2002 [30], the authors gave a more specific approach – they develop a system named GLUE. Similar to the previous approach, we must give the similarity definitions. We can apply the notion of the joint probability distribution between any two concepts in this approach if we make the assumption that each concept is modeled as a set of instances. There are four probabilities in the distribution: $P(A,B)$, $P(A,\neg B)$, $P(\neg A,B)$, and $P(\neg A,\neg B)$. Suppose an instance is randomly chosen from the universe, $P(A,B)$ is the probability that the instance belongs to A and B, $P(A,\neg B)$ is the probability that the instance belongs to A but not to B, $P(\neg A,B)$ is the probability that the instance belongs to B but not to A, and $P(\neg A,\neg B)$ is the probability that the instance belongs to neither A nor B. There are two similarity functions depending on different cases. In most cases, we can use (1), called the Jaccard coefficient [30], as the similarity measure.

$$Jaccardsim(A, B) = \frac{P(A \cap B)}{P(A \cup B)} = \frac{P(A, B)}{P(A, B) + P(A, \neg B) + P(\neg A, B)} \quad (1)$$

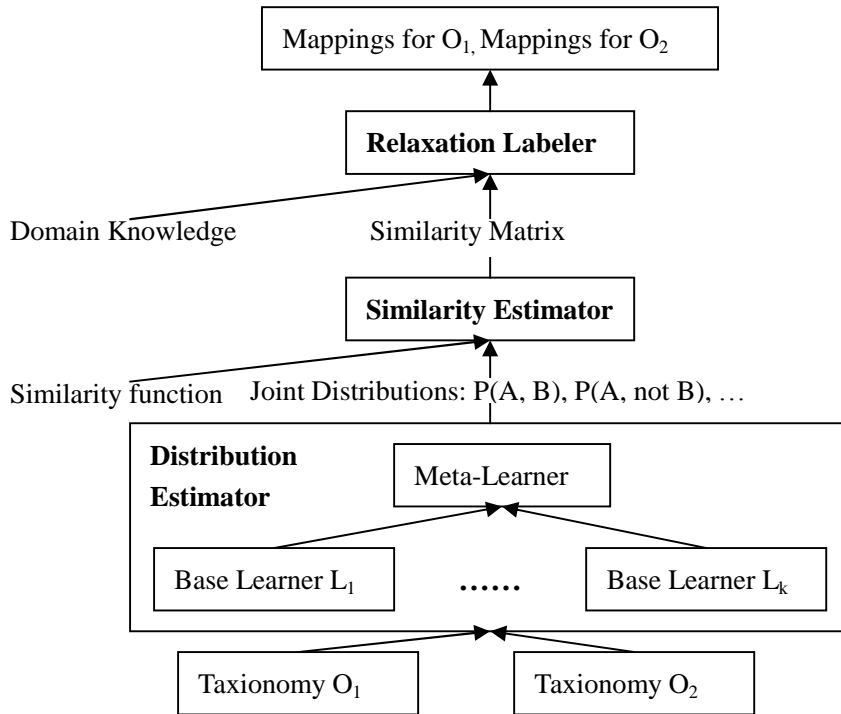


Figure 2.6.3.3: The GLUE Architecture

(Modified from “Learning to Map between Ontologies on the Semantic Web” [30])

Figure 2.6.3.3 shows the architecture of this system. The aim of the Distribution Estimator is to compute the probability distribution. It accepts two taxonomies O_1 and O_2 as input and compute the joint probability distribution for every pair of concepts (belong to either ontology respectively). The Distribution Estimator contains a set of base learners and a meta-learner. There are two types of base learners: content learner and name learner.

In this system, every instance has a name and a set of attributes, and the instance together with its attributes is regarded as the “textual content” [30] of the instance. The content learner uses the Naïve Bayes learning technique to categorize the textual content of the instance. In order to make a prediction, the content learner computes the probability that an input instance is an instance of a certain category, given its tokens. So let $d = \{w_1, \dots, w_k\}$ be the textual content of an instance and w_1, \dots, w_k are its tokens. We need to compute $P(A | d)$ which can be rewritten as:

$$\frac{P(d | A)P(A)}{P(d)} \quad (2)$$

where $P(A)$ can be computed as the portion of the training instances belong to A , and $P(d)$

here can be skipped since it is a normalizing constant. $P(d | A)$ can be computed as

$$P(d | A) = P(w_1 | A)P(w_2 | A) \cdots P(w_k | A) \quad (3)$$

$$P(w_i | A) = \frac{n(w_i | A)}{n(A)} \quad (4)$$

where $n(A)$ is the total number of token positions of **all** training instances that belong to A, and $n(w_i | A)$ computes the frequency of w_i (the number of times w_i appears in all training instances that belong to A). $P(\neg A, B)$ can be computed in a similar way. According to the authors of this paper, the content learner can get a very good result when the size of textual elements is large [30].

The name learner works the similar way with the content learner. The difference from the content learner is that the name learner uses the full name of the input instance which is the concatenation of concept names from the root to itself in the taxonomy tree. The meta-learner combines the predication produced by the base learners by a weighted sum. The Similarity Estimator is a simple layer where a similarity function is applied and output a similarity matrix. The aim of the Relaxation Labeler is to seek the mapping configuration that best fits the domain knowledge after taking the output of the Similarity Estimator.

2.7 Similarity Measures

As we state in previous sections, similarity measures play a very significant role in ontology matching. Apart from the measures above, Alexander Maedche and Steffen Staab [35] present a set of similarity measures which measures similarity between ontologies at two semiotic levels – the lexical level and the conceptual level. It is different from the approaches we discuss above. The previous approaches use the formal structures of ontologies and match the concept nodes. However, all ontologies in real-world not only specify the conceptualization by logical structures, but also refer to terms restricted by human natural languages use. So this approach measures the similarity at two different levels.

Definition (Lexicon) [35]: The lexicon (X) is composed of a set of concept terms (X^c) and relation terms (X^r). $X = X^c \cup X^r$

Definition (Core Ontology) [35]: A core ontology (O) is a tuple (A, P, D, X, F, G) where A is a set of concept symbols, P is a set of relation symbols, D is a set of statements, L is a lexicon and F, G are two reference functions. G is the reference function that links set of lexical entries to the set of relations they refer to, and F will link to the set of concepts they refer to.

2.7.1 Lexical Comparison Level

A lexical similarity measure for strings is defined as follows:

$$SM(L_i, L_j) = \max\left(0, \frac{\min(|L_i|, |L_j|) - ed(L_i, L_j)}{\min(|L_i|, |L_j|)}\right) \in [0,1] \quad (5)$$

where L_i, L_j are two lexical entries, $ed(L_i, L_j)$ is the edit distance [35] which is used to weight the difference between two strings. Edit distance computes the minimum number of token required to transform one string into another by means of insertions, deletions, and substitutions, e.g. the edit distance between “MScThesis” and “MSc_Thesis” is 1 since it takes one step (insert a character “_”) to finish the transformation. The result computed by the above formula falls in the range [0,1] where 1 represents good and 0 means bad. As for the lexical similarity measure for concepts in ontologies, the formula can be regard as the average of string matching:

$$\overline{SM}(X_1, X_2) = \frac{1}{|X_1|} \sum_{L_i \in X_1} \max_{L_j \in X_2} SM(L_i, L_j) \quad (6)$$

where X is the lexicon that consists a set of lexical entries, $X = X^c \cup X^r$, X^c is for concepts,

X^r is for relations, X_1 and X_2 are two this kind of lexical sets of two ontologies.

$\overline{SM}(X_1, X_2)$ is an asymmetric measure, i.e. $\overline{SM}(X_1, X_2)$ is different from $\overline{SM}(X_2, X_1)$.

2.7.2 Conceptual Comparison Level

a) Comparing taxonomies

There many approaches that compare the similarity of the two concepts between taxonomies, but few of these approaches compare two **taxonomies** themselves. Given two concepts C_1 and

C_2 from two taxonomies H_1 and H_2 , and a lexical entry $L \in X_1^c \cap X_2^c$ that refer to C_1 and

C_2 via two reference functions F_1 and F_2 , then the intensional semantics of $C_1(C_2)$ are composed of “semantic cotopy” (SC) of $C_1(C_2)$.

$$SC(C_i, H) = \{C_j \in A \mid H(C_i, C_j) \vee H(C_j, C_i)\} \quad (7)$$

where H is the taxonomy, A is a set of concept symbols of the ontology. This formula can be extended to sets as follows:

$$SC(\{C_1, \dots, C_n\}, H) = \bigcup_{i=1, \dots, n} SC(C_i, H) \quad (8)$$

The taxonomic overlap (TO) between taxonomy H_1 and H_2 can be computed with the formula:

$$TO'(L, O_1, O_2) = \frac{|F_1^{-1}(SC(F(\{L\}), H_1)) \cap F_2^{-1}(SC(F(\{L\}), H_2))|}{|F_1^{-1}(SC(F(\{L\}), H_1)) \cup F_2^{-1}(SC(F(\{L\}), H_2))|} \quad (9)$$

where O_1 and O_2 are two ontologies, F is the reference function that links set of lexical entries to the set of concepts they refer to. But there is another case: $L \in X_1^c$ and $L \notin X_2^c$. In this case, the taxonomic overlap can be computed as follows:

$$TO''(L, O_1, O_2) = \max_{C \in C_2} \left\{ \frac{|F_1^{-1}(SC(F(\{L\}), H_1)) \cap F_2^{-1}(SC(F(\{L\}), H_2))|}{|F_1^{-1}(SC(F(\{L\}), H_1)) \cup F_2^{-1}(SC(F(\{L\}), H_2))|} \right\} \quad (10)$$

Now we can get the average similarity for taxonomies:

$$\overline{TO}(O_1, O_2) = \frac{1}{X_1^c} \sum_{L \in X_1^c} TO(L, O_1, O_2) \quad (11)$$

which is constrained with

$$TO(L, O_1, O_2) = \begin{cases} TO'(L, O_1, O_2) & \text{if } L \in X_2^c \\ TO''(L, O_1, O_2) & \text{if } L \notin X_2^c \end{cases} \quad (12)$$

b) Comparing relations

Similar to the above comparison, we can compute the relation overlap (RO), i.e. the accuracy two relations match. RO can be computed based on the geometric mean value of similarity their domain concepts. Similar to taxonomies comparison, we define the upwards cotopy (UC) in order to consider the similarity of concepts:

$$UC(C_i, H) = \{C_j \in A | H(C_i, C_j)\} \quad (13)$$

Then, similar to the definition of TO' , the concept match is defined as (14) shows:

$$CM(C_1, O_1, C_2, O_2) = \frac{|F_1^{-1}(UC(C_1, H_1)) \cap F_2^{-1}(UC(C_2, H_2))|}{|F_1^{-1}(UC(C_1, H_1)) \cup F_2^{-1}(UC(C_2, H_2))|} \quad (14)$$

RO' of relations R_1 and R_2 can be defined as (15).

$$RO'(R_1, O_1, R_2, O_2) = \sqrt{CM(d(R_1), O_1, d(R_2), O_2) \cdot CM(r(R_1), O_1, r(R_2), O_2)} \quad (15)$$

Consider $L \in X_2^r, L \in X_1^r$

$$RO''(L, O_1, O_2) = \frac{1}{|G_1(\{L\})|} \sum_{R_1 \in G_1(\{L\})} \max_{R_2 \in G_2(\{L\})} \{RO'(R_1, O_1, R_2, O_2)\} \quad (16)$$

$$RO'''(L, O_1, O_2) = \frac{1}{|G_1(\{L\})|} \sum_{R_1 \in G_1(\{L\})} \max_{R_2 \in P_2} \{RO'(R_1, O_1, R_2, O_2)\} \quad (17)$$

where R_1 and R_2 are two relations we want to compare. G is the reference function that links set of lexical entries to the set of relations they refer to. Since there are several different

conditions, the author gives (15), (16) and (17) respectively. Then with combination of the above definitions, we get RO for $L \in X_1^c$

$$RO(L, O_1, O_2) = \begin{cases} RO''(L, O_1, O_2) & \text{if } L \in X_2^r \\ RO'''(L, O_1, O_2) & \text{if } L \notin X_2^r \end{cases} \quad (18)$$

Similar to taxonomies, the average relation overlap is then defined grounded with the condition (18):

$$\overline{RO}(O_1, O_2) = \frac{1}{|X_1^r|} \sum_{L \in X_1^r} RO(L, O_1, O_2) \quad (19)$$

Now we can see this method is more complex than the previous approaches, but it is very interesting and creative.

2.8 Toolkits

2.8.1 Jena

HP has developed a toolkit named Jena for developing applications within the Semantic Web. The toolkit contains a RDF/XML Parser (ARP), Jena relational database interface, integrated query language (RDQL), a server for publishing RDF models on the web (Joseki [27]) and a set of Java API for RDF. It has added support for DAML+OIL since version 1.2.

2.8.2 Sesame

Sesame [34] is an open source project which is an RDF Schema-based repository. It has several useful features:

- a) Data administration: add, delete data.
- b) Export: exports the data in repository to RDF documents.
- c) RQL engine: can be used to evaluate RQL queries.

2.8.3 DAMLJessKB

DAMLJessKB [32] is set of Java API for reasoning with DAML within the Semantic Web. It is built based on Jena and Jess [31]. It now supports DAML, RDFS, RDF and XML Schemas. It provides abilities such as reading DAML files, converting the information to the DAML language, and making queries.

Part II

3. Project Plan

3.1. Aims and Objectives

The main aim of this project is to develop a search engine based on ontology matching within the Semantic Web. Since the Semantic Web is distributive, there are a lot of resource descriptions where two concepts within different ontologies are equivalent, but they are described in different terms. This project is to match those elements, and then bring a more accuracy search result. This project can be divided into several sub-tasks.

- a. A **DAML builder and converter**: This module is to covert data file into Semantic Web (SW) form, and to generate the SW data file based on the certain ontology semi-automatically.
- b. A **DAML crawler** (optional): This agent is to travel the web and collect the DAML content on the Web.
- c. A **query builder**: A query builder is the bridge between the user and machine learning agent. It accepts the input of the user and constructs a kind of “query language”. Then it transfers the query to machine learning agent. It has a friendly interface that will also ease the user input.
- d. A **machine learning agent**: This module is the core of this project. Some machine learning techniques involve this project in this layer. The agent will measure the similarity of the elements and determine if they are equivalent. Then it will transfer the output to a CGI-like program to return the result to the user.

3.2. Initial Design and Specification

Since a detailed design and specification of this project is not presented now, I can only give a rough illustration. I don't guarantee that all features here will be implemented and some components may be changed, added or removed. The project has 3 main sub-tasks:

3.2.1 DAML Converter and Builder

I will prefer DAML to RDF as the semantic web data form in this project since there are many data sets that open to public. Data will be stored in a relational database since the response time is very important to a search engine. I will also use some artificial data if the data on the Web is not suitable for my project. In this particular case, I will make an agent that

will generate data in SW (Semantic Web) form automatically and convert data in Non-SW form to SW form. This agent will be implemented by java with some DAML API. It has a GUI user interface, and I will also implement a web interface if time is enough.

3.2.2 Query Builder

The aim of query builder is to ease the process of user input, especially when a user wants to make a complex query. It will take the user requests and format them into a kind of “Query Language” to agents. This part will be a cgi-like program, built with some jsp pages or servlets.

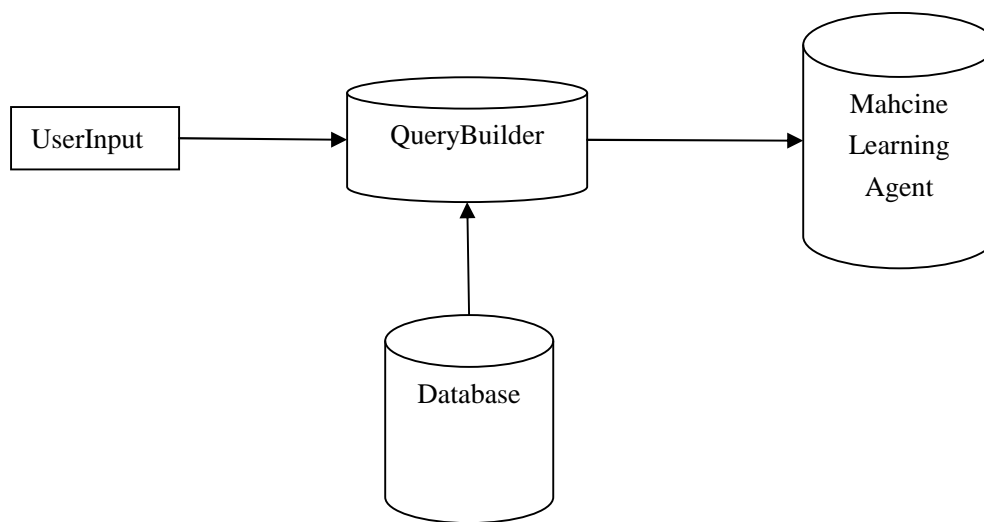


Figure 3.2.2.1 Query Builder

3.2.3 Machine Learning Agent

This is the core of this project. The agent will accept the queries, match the contents against the requests and measure the similarity among elements of two ontologies and determine if they are matched candidates. Then it will bring the information back to the user.

Some machine learning techniques such as Naïve Bayes, involve in this layer. As we mentioned in previous sections, there are several types of mapping. I will focus on one-to-one mapping in this project, rather than complex types. Complex mapping may be dealt with if there is time left, e.g. “Full Name” maps to the combination of “First Name” and “Last Name”.

As for the similarity measures, I have not decide what type of measures I should use. This is because almost all methods are not suitable for every domain or condition. So I will compare several methods when developing and choose one that best fits my data sets, or may

implement a hybrid system.

3.2.4 DAML Crawler

DAML crawler is a program that collects DAML statements by traveling on the Web. The main aim of the crawler is to collect DAML content on the Web periodically. This part is optional since there are several existing crawlers on the Web and I can also use the existing tools.

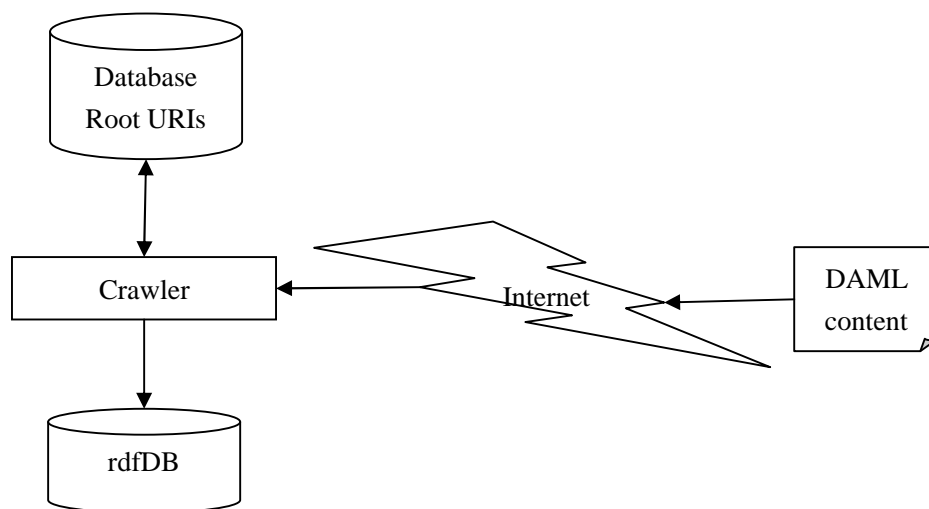
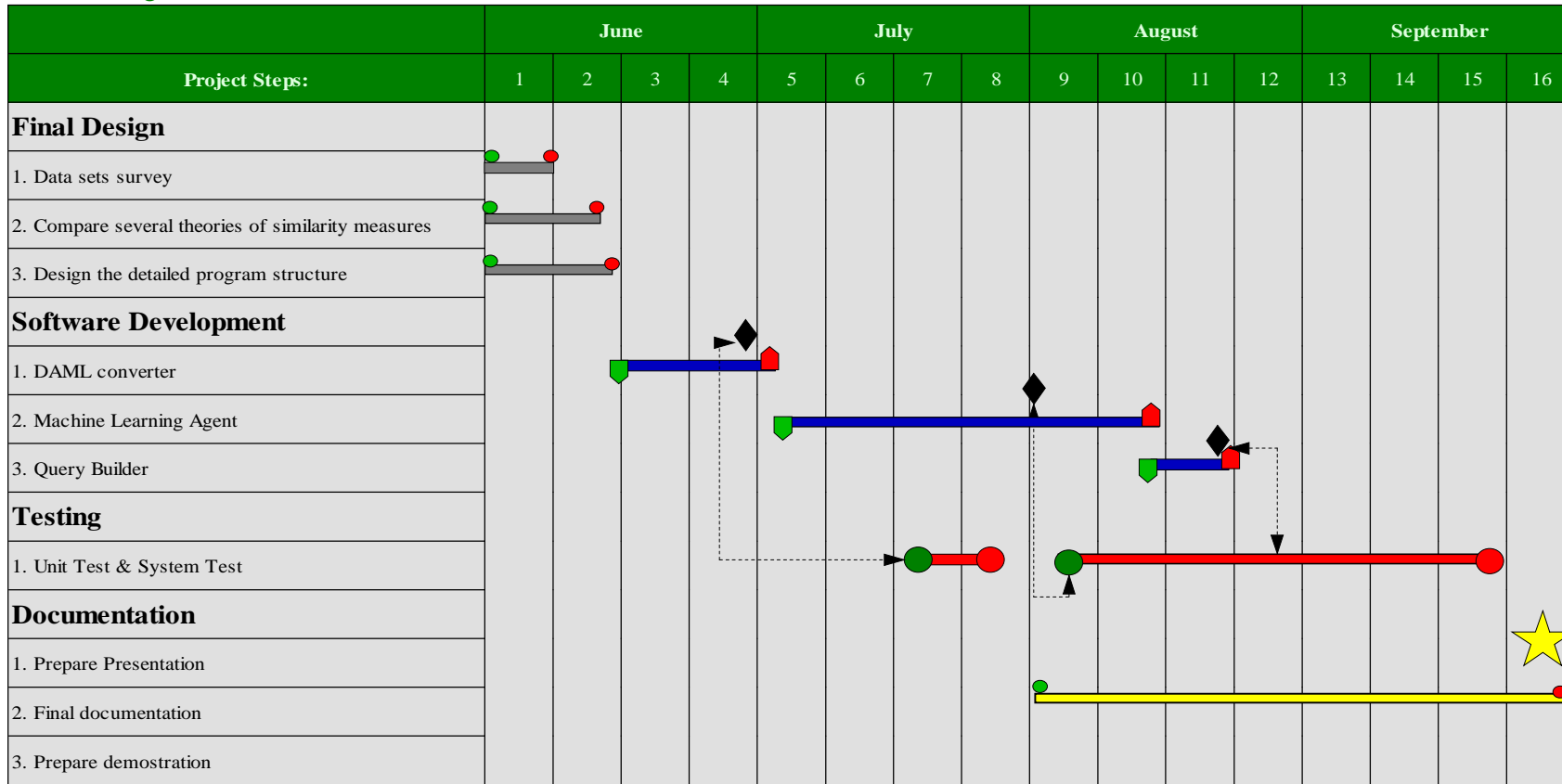


Figure 3.2.4.1 DAML Crawler

3.3 Project Schedule



Part III

4. Bibliography

1. The Semantic Web - ISWC 2002: First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002 : proceedings / Ian Horrocks, James Hendler (eds.)
2. Advances in web-based learning: first international conference, ICWL 2002, Hong Kong, China, August 17-19, 2002 : proceedings / Joseph Fong ... [et al.] (eds.)

5. References

- [1] W3C Semantic Web, <http://www.w3.org/2001/sw/>
- [2] Tim Berners-Lee, James Hendler, Ora Lassila, The Semantic Web, Scientific American, <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21>
- [3] Sean B. Palmer, The Semantic Web: An Introduction, 2001, <http://infomesh.net/2001/swintro/>
- [4] Eric Prudhommeaux, Presentation of W3C and Semantic Web, 2001, <http://www.w3.org/2001/Talks/0710-ep-grid>
- [5] Tim Berners-Lee, W3C Naming and Addressing Overview (URIs. URLs, ...), <http://www.w3.org/Addressing/>
- [6] W3C, W3C Resource Description Framework (RDF), <http://www.w3.org/RDF/>
- [7] The DARPA Agent Markup Language (DAML), <http://www.daml.org/>
- [8] AQ-Search Group (Jianghua Tu, Rui Feng, Zhuoyun Li, Wei Tong, Zaiqiang Liu and Instructor --- Prof. Kokar), AQ-Search Project, 2002
- [9] Aaron Swartz, The Semantic Web (for Web Developers), May 2001, <http://logicerror.com/semanticWeb-webdev>
- [10] Aaron Swartz, The Semantic Web In Breadth, May 2002, <http://logicerror.com/semanticWeb-long#acks>

-
- [12] Sandro Hawke, How the Semantic Web Works, April 2002, <http://www.w3.org/2002/03/semweb/>
- [13] W3C, HTTP Specifications and Drafts, <http://www.w3.org/Protocols/Specs.html>
- [14] IETF, Hypertext Transfer Protocol -- HTTP/1.1 - Draft Standard RFC 2616, <http://www.ietf.org/rfc/rfc2616.txt>
- [15] W3C, Extensible Markup Language (XML), <http://www.w3.org/XML/>
- [16] W3C, Extensible Markup Language (XML) 1.0 (Second Edition), 6 October 2000, <http://www.w3.org/TR/REC-xml>
- [17] Erik T. Ray, Learning XML, First Edition, January 2001, ISBN: 0-59600-046-4, 368 pages.
- [18] R. Allen Wyke, Brad Leupen, Sultan Rehman, XML Programming, 2002, Microsoft Press
- [19] W3Schools (<http://www.w3schools.com>), XML Schema Tutorial, (2001) http://www.w3schools.com/schema/schema_intro.asp
- [20] Tim Bray, What is RDF? <http://www.xml.com/lpt/a/2001/01/24/rdf.html>
- [21] W3C, RDF Primer, W3C Working Draft 23 January 2003, <http://www.w3.org/TR/2003/WD-rdf-primer-20030123/>
- [22] W3C, Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999, <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222>
- [23] W3C, RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft 23 January 2003, <http://www.w3.org/TR/rdf-schema/>
- [24] Roxane Ouellet, Uche Ogbuji, Introduction to DAML: Part I (2002), <http://www.xml.com/lpt/a/2002/01/30/daml1.html>
- [25] Roxane Ouellet, Uche Ogbuji, Introduction to DAML: Part II (2002), <http://www.xml.com/lpt/a/2002/03/13/daml.html>
- [26] Adam Pease, Why Use DAML? (10 April, 2002), <http://www.daml.org/2002/04/why.html>
- [27] <http://www.joseki.org/>

-
- [28] Jena, <http://www.hpl.hp.com/semweb/>
- [29] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199-220, 1993.
- [30] AnHai Doan, Jayant Madhavan, Pedro Domingos, Alon Halevy. Learning to Map Between Ontologies on the Semantic Web. May 2002. <http://www2002.org/CDROM/refereed/232/>
- [31] Jess, <http://herzberg.ca.sandia.gov/jess/>
- [32] DAMLJessKB, <http://edge.mcs.drexel.edu/assemblies/software/damljesskb>
- [33] Xiaomeng Su, A Text Categorization Perspective for Ontology Mapping, <http://www.idi.ntnu.no/~xiaomeng/paper/Position.pdf>
- [34] Sesame, <http://sesame.aidministrator.nl/>
- [35] Alexander Maedche and Steffen Staab, Comparing Ontologies — Similarity Measures and a Comparison Study, March 2001, <http://www.aifb.uni-karlsruhe.de/~sst/Research/Publications/report-aifb-408.pdf>