OntoStore: Ontology-Driven Information Extraction for Semantic Annotation of the Web

Authors

Md. Nizam Uddin Samrat Student Id: 084422
Tanvir Ahmed Student Id: 084442

Supervisor

Mahmud Hasan
Assistant Professor
Department of Computer Science & Engineering (CSE)
Islamic University of Technology (IUT)

A Thesis submitted to the Department of Computer Science and Engineering (CSE) in Partial Fulfillment of the requirements for the degree of Bachelor of Science in CSE (Computer Science & Engineering)

Department of Computer Science & Information Technology (CIT)
Islamic University of Technology (IUT)
Organization of the Islamic Cooperation (OIC)
Gazipur, Bangladesh.
September, 2012

CERTIFICATE OF RESEARCH

This is to certify that the work presented in this thesis paper is the outcome of the research carried out by the candidates under the supervision of **Mr. Mahmud Hasan, Assistant professor, Department of Computer Science and Engineering, IUT, Gazipur.** It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree or any judgment.

Authors ~

Md. Nizam Uddin Samrat **Tanvir Ahmed**

Signature of Supervisor ~

Mahmud Hasan
Assistant professor
Department of CSE, IUT

Signature of the Head of the Department ~

Prof. Dr. M. A. Mottalib
Head, Department of CSE, IUT

Abstract

Automated annotation of web pages is required for the successful implementation of Semantic Web. OntoStore is a new ontology-driven domain-independent approach which aims to provide a platform for the operation of semantic applications. The prototype of OntoStore is fully functional and this paper explains the process of OntoStore, how it operates, instance extraction and verification using OntoStore and a comparison with existing systems.

Index Terms—Semantic Web, Semantic annotation, Information extraction, Ontology-driven, Domain-independent.

Acknowledgement

At the very beginning we express our heartiest gratitude to Almighty Allah for His divine blessings which allowed us to bring this research work to life.

We are grateful and indebted to our supervisor **Mr. Mahmud Hasan,** Assistant Professor, Department of Computer Science and Engineering, IUT. His supervision, knowledge and relentless support have time and again proved to be invaluable and allowed us to complete this endeavor successfully. Their patience and encouragement allow us to stand where we stand today.

Our appreciation extends to all the respected faculty members of the Department of Computer Science & Engineering especially **Dr. Kamrul Hasan** for the assistance they have provided to us.

We are also grateful to **Luke K. McDowell**, Associate Professor, Department of Computer Science, United States Naval Academy and **Michael Cafarella**, assistant professor, Computer Science and Engineering, University of Michigan for their helpful suggestions. We are also thankful to **Google™** for using their web service API.

Finally we would like to extend our gratitude to our batch mates, students, staffs and everyone else who have contributed to this work in their own way.

Table of Contents

Dedicated to our loving parents.

Chapter 1

Introduction

1.1 Overview

The Internet and the World Wide Web have brought a revolution to information technology and the daily lives of most people. Considering the structure of the World Wide Web we can define it as Syntactic Web. The Syntactic Web is a place where computers do only the presentation and people do the linking and interpreting. Then the question arises "Why not get computers to do more of the hard work"?

Some examples of hard works using the Syntactic Web may include:

- Complex queries involving background knowledge like finding information about animals that use sonar but are not bats, dolphins or whales.
- Locating information in data repositories like travel enquiries, prices of goods and services or results of human genome experiments.
- Delegating complex tasks to web "agents" like booking a holiday next weekend somewhere warm, not too far away, and where they speak Bengali or English.

1.2 Problem Statement

1.2.1 Structure of Html Documents

Currently the Syntactic Web is based mainly on documents written in HTML. Metadata tags provide a method by which computers can categorize the content of web pages. These elements can be used to define relationships for the enclosing HTML files only. Considering a typical web page, semantic contents of the page are accessible to the humans but not to the computers. And that is the reason why computers cannot perform the hard tasks described above.

1.2.2 Semantic Web – The ultimate solution

The word "semantic" stands for the "meaning of". The semantic of something is the meaning of something. Therefore we can say that, the Semantic Web is a Web with a meaning. The Semantic Web is a web that is able to describe things in a way so that computer applications can understand.

This is what the Semantic Web is all about. The Semantic Web is not about links between web pages. Rather it describes the relationship between the things in the web.

1.2.3 Purpose of Semantic Web

The main purpose of the Semantic Web is to enable users to find, share, and combine information more easily with less intervention. Humans are capable of using the Web to perform complex tasks but machines cannot accomplish all of these tasks without human directions as because web pages are designed to be read by people, not machines. The semantic web is a vision of information that can be readily interpreted by machines, so machines can perform more of the tedious work involved in finding, combining, and acting upon information on the web.

1.2.4 Operation of Semantic Web

Statements are built with syntax rules. The syntax of a language defines the rules for building the language statements. But how can syntax become semantic and understandable by computer applications? If we consider the following statements:

- Iron Maiden is a popular band from England.
- Nicko McBrain plays drums in Iron Maiden.
- "Fear of the dark" is the most popular song by Iron Maiden.

Sentences like the ones above can be understood by people. But how can they be understood by computers?

Semantic web uses technologies like RDF (Resource Description Framework), OWL (Web Ontology Language) and XML (Extensible Markup Language) which can describe arbitrary things such as people, meetings, or airplane parts. These technologies are combined in order to provide descriptions that supplement or replace the content of Web documents. Thus, content may manifest itself as descriptive data stored in Web-accessible databases. The machine-readable descriptions enable content managers to add meaning to the content to describe the structure of the knowledge we have about that content. In this way, a machine can process knowledge itself, instead of text, using processes similar to human deductive reasoning and inference, thereby obtaining more meaningful results and helping computers to perform automated information gathering and research.

1.2.5 Semantic Web Requirements – Semantic Annotation

The availability of web pages with proper semantic annotations is the first requirement for the successful implementation of semantic web. Without the existence of annotated web pages it is impossible to achieve semantic web. For this reason ontologists have developed different ontologies so that web pages can be annotated with metadata in a systematic way. Usually there are two ways of annotating web pages with metadata. First, we can provide the metadata manually to annotate the page while developing it. Second, we can annotate the web pages in an automatic or a semi-automatic way. The problem associated with the first approach is for the traditional developers of web pages it is difficult to understand the concept of ontologies and the other requirements of semantic web. As developing web pages with semantic annotation does not

provide any benefit immediately it is difficult to motivate them. So they need tools like EasyRDF, Quest, BigData etc. that will help them to annotate the pages while developing. But the question remains, what will happen to the millions of pages existing in the web currently? For this reason automated annotation of web pages is required.

1.3 Research Challenges

In this paper we introduce a new ontology-driven domain-independent approach OntoStore, which aims to extract the instances of an ontology form the web and tries to generate instances for that ontology automatically based on the requirements of a semantic application.

1.4 Motivations

We motivation is to create a platform for semantic applications. We intend to accept requests for ontologies from various semantic application developers and we aim to extract instances of those ontologies from the web. We want to implement the annotation process by enriching the ontologies based on the extracted instances. We let the semantic applications access the modified ontology.

1.5 Thesis Outline

In Chapter 1 a brief introduction of the study has been provided. Chapter 2 contains a brief description of some of the Semantic Web technologies. Chapter 3 elaborates on the proposed methodology stating how OntoStore operates to extract the instances of the ontology classes. Chapter 4 shows the results that were achieved implementing the proposed approach for implementing the process. Finally Chapter 5 briefly states the possible future works and concludes the paper.

Chapter 2

Semantic Web Technologies

2.1 Ontology

2.1.1 What is Ontology?

In a widely-quoted definition, ontology is a specification of a conceptualization. Ontology allows a programmer to specify, in an open, meaningful, way the concepts and relationships that collectively characterize some domain. For example the *wine ontology* was developed initially for a particular application, such as a stock-control system at a wine warehouse. Ontology may be considered similar to a well-defined database schema. The advantage to ontology is that it is an explicit, first-class description. So having been developed for one purpose, it can be published and reused for other purposes. For example, a given winery may use the wine ontology to link its production schedule to the stock system at the wine warehouse. Alternatively, a wine recommendation program may use the wine ontology, and a description (ontology) of different dishes to recommend wines for a given menu.

2.1.2 How to define Ontology

There are many ways of writing down ontology, and a variety of opinions as to what kinds of definition should go in one. In practice, the contents of ontology are largely driven by the kinds of application it will be used to support. RDFS is the weakest ontology language to write down ontology. RDFS allows the ontologist to build a simple hierarchy of concepts, and a hierarchy of properties. Let us consider the following trivial characterization in figure 1.

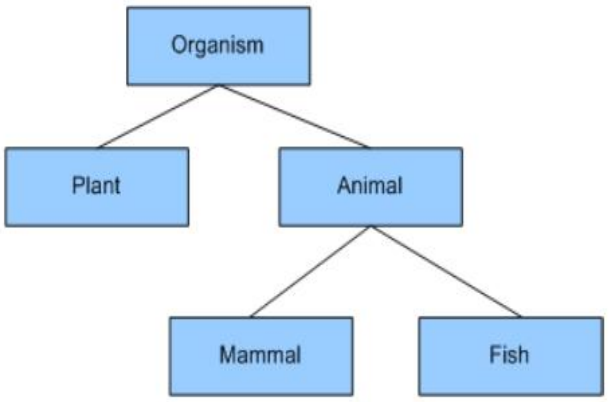

Figure-1: A simple concept hierarchy

Using RDFS, we can say that this ontology has five *classes*, and Plant is a *sub-class of* Organism and so on. So every animal is also an organism. A good way to think of these classes is as

describing sets of *individuals*: organism is intended to describe a set of living things, some of which are animals (i.e. a sub-set of the set of organisms is the set of animals), and some animals are fish (a subset of the set of all animals is the set of all fish).

To describe the attributes of these classes, we can associate *properties* with the classes. For example, animals have sensory organs (noses, eyes, etc.). A general property of an animal might be senseOrgan, to denote any given sensory organs a particular animal has. In general, fish have eyes, so a fish might have a eyes property to refer to a description of the particular eye structure of some species. Since eyes are a type of sensory organ, we can capture this relationship between these properties by saying that eye is a sub-property-of senseOrgan. Thus if a given fish has two eyes, it also has two sense organs. (It may have more, but we know that it must have two).

We can describe this simple hierarchy with RDFS. In general, the class hierarchy is a graph rather than a tree (i.e. not like Java class inheritance). The slime mold is popularly, though perhaps not accurately, thought of as an organism that has characteristics of both plants and animals. We might model a slime mold in this ontology as a class that has both plant and animal classes among its super-classes. RDFS is too weak a language to express that a thing cannot be both a plant and an animal (which is perhaps lucky for the slime molds). In RDFS, we can only name the classes, We cannot construct expressions to describe interesting classes. However, for many applications it is sufficient to state the basic vocabulary, and RDFS is perfectly well suited to this.

Note also that we can both describe classes, in general terms, and we can describe particular *instances* of those classes. So there may be a particular individual Fred who is a Fish (i.e. has rdf:type Fish), and who has two eyes. His companion Freda, a Mexican Tetra, or blind cave fish, has no eyes. One use of an ontology is to allow us to fill-in missing information about individuals. Thus, though it is not stated directly, we can deduce that Fred is also an Animal and an Organism. Assume that there was no rdf:type asserting that Freda is a Fish. We may still infer Freda's rdf:type since Freda has lateral lines as sense organs, and these only occur in fish. In RDFS, we state that the *domain* of the lateralLines property is the Fish class, so an RDFS reasoner can infer that Freda must be a fish.

2.1.3 Ontology and Reasoning

One of the main reasons for building an ontology-based application is to use a reasoner to derive additional truths about the concepts you are modeling. We saw a simple instance of this above: the assertion "Fred is a Fish" *entails* the deduction "Fred is an Animal". There are many different styles of automated reasoner, and very many different reasoning algorithms.

2.2 Resource Description Framework (RDF)

Semantic Web uses the RDF (Resource Description Framework) for describing information and resources on the web. Putting information into RDF files, makes it possible for web applications ("web spiders") to search, discover, pick up, collect, analyze and process information from the web. If information about music, cars, tickets, etc. were stored in RDF files, intelligent web applications could collect information from many different sources, combine information, and present it to users in a meaningful way.

RDF is best thought of in the form of node and arc diagrams.

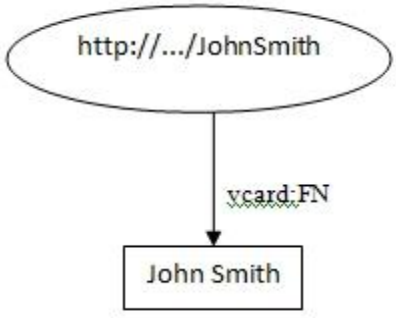

Figure-2: A simple RDF

The ***resource***, John Smith, is shown as an **eclipse** and is identified by a Uniform Resource Identifier (URI), in this case "http://.../JohnSmith".

Resources have ***properties***. The figure shows only one property, John Smith's full name. A property is represented by an **arc**, labeled with the name of a property. The name of a property is also a URI, but as URI's are rather long and cumbersome, the diagram shows it in XML qname form. The part before the ':' is called a namespace prefix and represents a namespace. The part after the ':' is called a local name and represents a name in that namespace. Properties are identified by a URI. The nsprefix:localname form is a shorthand for the URI of the namespace concatenated with the localname.

Each property has a value. In this case the value is a ***literal***, which for now we can think of as a strings of characters, Literals are shown in **rectangles**.

In the first example, the property value was a literal. RDF properties can also take other resources as their value. Using a common RDF technique, this example shows how to represent the different parts of John Smith's name:

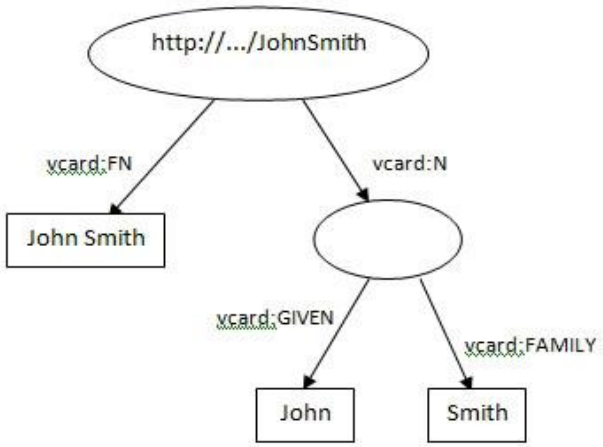

Figure-3: A RDF with a resource as a property

Each arc in an RDF Model is called a ***statement***. Each statement asserts a fact about a resource. A statement has three parts:

- 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 pointed to by the arc.

A statement is sometimes called a **triple**, because of its three parts. The statements in figure-2 will generate a RDF file like the following :

```
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#'
 >
  <rdf:Description rdf:about='http://…/JohnSmith'>
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:nodeID="A0"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A0">
    <vcard:Given>John</vcard:Given>
    <vcard:Family>Smith</vcard:Family>
  </rdf:Description>
</rdf:RDF>
```

RDF is usually embedded in an <rdf:RDF> element. The RDF element defines the two namespaces used in the document. There is then an <rdf:Description> element which describes the resource whose URI is "http://.../JohnSmith". If the rdf:about attribute was missing, this element would represent a blank node.

The <vcard:FN> element describes a property of the resource. The property name is the "FN" in the vcard namespace. RDF converts this to a URI reference by concatenating the URI reference

for the namespace prefix and "FN", the local name part of the name. This gives a URI reference of "http://www.w3.org/2001/vcard-rdf/3.0#FN". The value of the property is the literal "John Smith".

The <vcard:N> element is a resource. In this case the resource is represented by a relative URI reference. RDF converts this to an absolute URI reference by concatenating it with the base URI of the current document.

One thing to be noticed, the blank node in the Model has been given a URI reference. It is no longer blank.

2.3 Approaches towards implementation of Semantic annotation

The Semantic Web is not a very fast growing technology. One of the reasons for that is the learning curve. RDF was developed by people with academic background in logic and artificial intelligence. For traditional developers it is not very easy to understand. So far we have seen that applications that use semantic web technologies can be very promising and useful. But the success of these applications depends on the successful implementation of semantic web. And the success of semantic web depends largely on the existence of a sufficient amount of high-quality, relevant semantic data. But till now relatively little such content has emerged. So researchers have investigated systems to assist users with producing (or annotating) such content, as well as systems for automatically extracting semantic content from existing unstructured data sources such as web pages.

According to Alex Iskold, there are two main approaches to implement Semantic Web Annotation:

1) Bottom Up - involves embedding semantic annotations (meta-data) right into the data.
2) Top down - relies on analyzing existing information and producing semantic contents automatically.

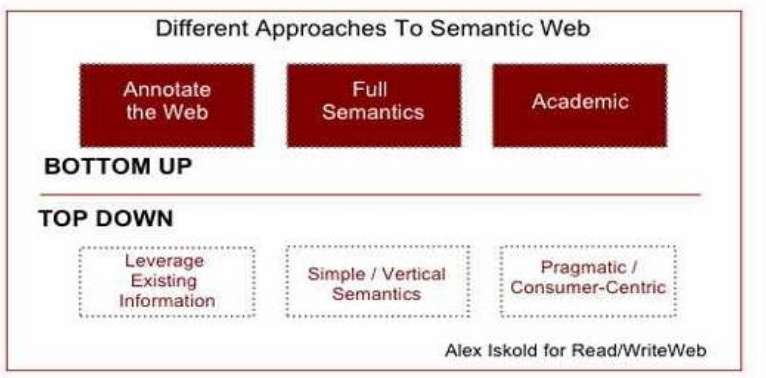

Figure-4: Different approaches to semantic web

In the bottom-up approach semantic annotations are embedded into the data in a manual or semi-automatic way. This annotation meant to be statically associated to the documents. Static annotation can:

(1) be incomplete or incorrect when the creator is not skilled enough;

(2) become obsolete, i.e. not be aligned with page updates;

(3) Be devious, e.g. for spamming or dishonest purposes; professional spammers could use manual annotation very effectively for their own purposes. For these reasons, Semantic Web needs automatic methods for page annotation.

Most systems for automated content generation or page annotation are given a small to moderate size of relevant data and the system sequentially processes each document. For each document, the system tries to extract relevant information and encode it using the predicates and classes of a given ontology. This extraction might utilize a domain-specific wrapper, constructed by hand or via machine learning techniques. More recent domain-independent approaches have utilized a named entity recognizer to identify interesting terms, and then used web searches to try to determine the term's class. In either case, these are document-driven systems whose workflow follows the documents.

Chapter 3

Proposed Method

3.1 The Process of OntoStore

OntoStore operates in an ontology-driven domain-independent manner. The objective of OntoStore is to provide a platform for the operation of the semantically-aware applications. It tries to extract the instances of a given set of ontologies that are required for the operation of a semantic application. When a semantic application requires service from OntoStore it can directly access the enriched ontologies to get the information required. The developer of a semantic application must submit the list of the ontologies to OntoStore before manufacturing so that OntoStore can search, store and annotate the instances of these ontologies and provide service as soon as the product is launched. The process of OntoStore is described below:

Input : OntSet = {A set of input ontologies}

Do{

Step 1 : O = SelectOntology(OntSet)

Step 2: C = TraverseOntologyClasses(O)

Do {

i) P1 = GeneratePattern(C, "null")

ii) IC+ = SearchCandidates(P1)

iii) P2 = GeneratePattern(C,IC)

iv) FC+ = SearchCandidates(P2)

}**While** Traverse is not complete

Step 3 : Instances = VarifyInstances(FC)

}**While** OntSet is not empty

Output : Annotated web pages for given ontologies

Figure-5: the working algorithm of OntoStore

Input: A set of ontologies.

Step 1: At step 1 OntoStore selects ontology from the given set of ontologies.

Step 2: OntoStore traverse through the classes(C) and subclasses of each ontology(O) in the input set in a bottom-up manner and searches for the instances of each ontology class.

Step 3: The instances are verified and stored for annotating of web pages.

Output: Annotated web pages.

The working algorithm of OntoStore is given in figure 5.

3.2 The Operation of OntoStore

Initially OntoStore takes a given set of ontologies and process them sequentially for storing the ontological instances into the RDF repository. First it selects an ontology(O) and traverse through all the classes(C) and subclasses of it for applying Hurst Phrases to generate some patterns(P1). Then it submits the pattern to Google™ for making a web search. Form the search result's abstract it selects the candidate instances. Next, it adds the candidates with the used patterns to generate new patterns(P2) which are also submitted to Google™ for another web search. From this search result we can get some candidate instances which are more reliable than the candidates of the previous search. After that the results are verified. Finally the annotation process is performed by enriching the ontology with the extracted instances.

Figure-6: The operation of OntoStore

When a semantic application requires service from OntoStore it can directly access the modified ontology to get the information required. If an application requires using a new ontology then the developer must submit the definition of the ontology to OntoStore before manufacturing the application so that OntoStore can search, annotate and store the resources of the new ontology and provide service as soon as the product is launched.

3.3 The Method of Instance Extraction

Initially OntoStore takes a given set of ontologies as input and process them sequentially for extracting the ontological instances. First, it selects an ontology(O) and traverse through all the classes(C) and subclasses of it in a bottom-up manner by applying Hearst Phrases[10] to generate some patterns(P1). Then, the pattern is submitted to Google™ for making a web search. Form the abstract of the search results it extracts the candidate instances(IC). Next, it adds the candidates with the patterns(P1) to generate new patterns(P2) which are also submitted to Google™ for another web search. From this search result we can get the final candidates(FC) which are more reliable than the candidates of the previous search. After that, the candidate instances(FC) are verified. The verification process is described in details in section 4. Finally the annotation process is performed by adding the verified instances in the appropriate classes of the ontology.

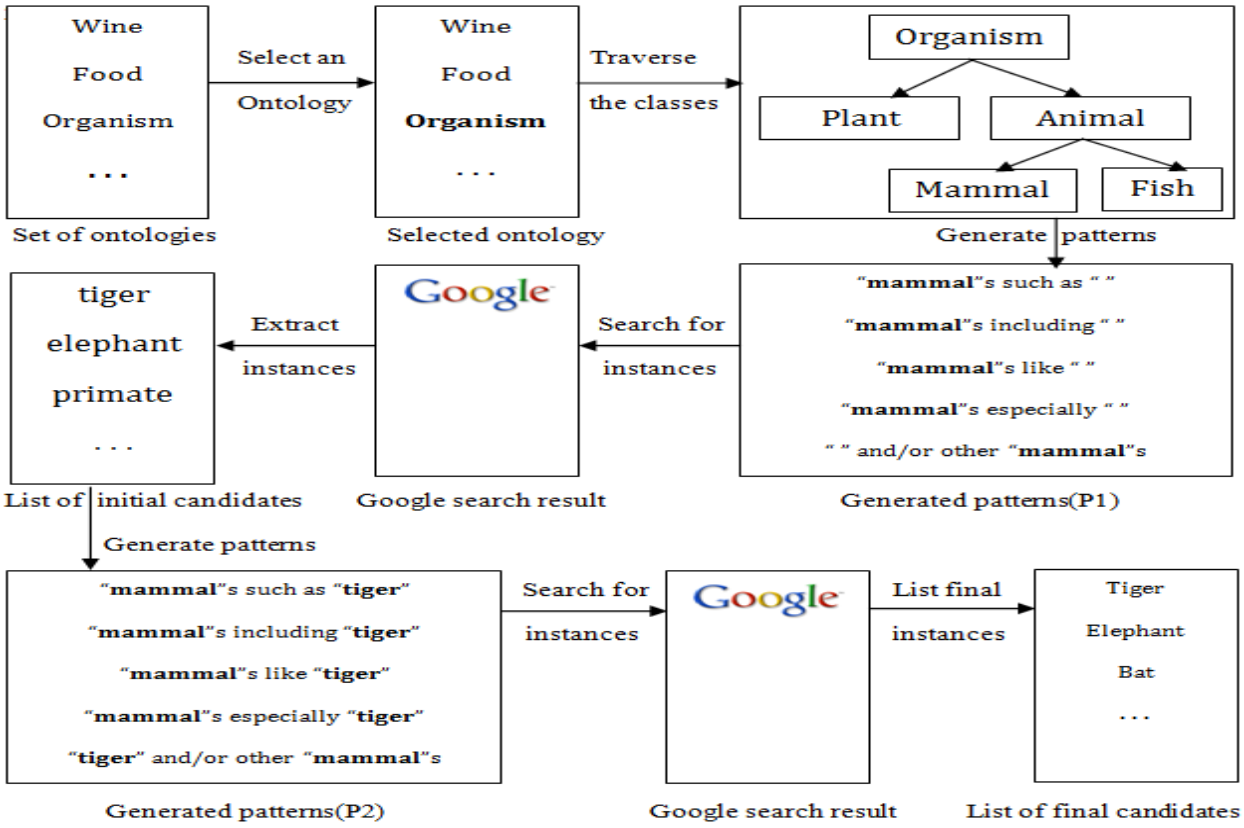

Figure-7: The method of instance extraction using OntoStore

Figure 2 shows the method of instance extraction using OntoStore. The ontology *Organism* is selected from the input ontology list and traversed in a bottom-up manner by OntoStore. Hearst Phrases[10] are added with ontology classes to generate patterns(P1) like “mammals such as”, “mammals including”, “mammals like”, “mammals especially”, “and/or other mammals” etc. These patterns are submitted to perform a Google query and a list of initial candidate(IC) instances are generated by extracting instances from the Google abstract. These candidate instances are added with the existing patterns to generate new patterns(P2) like “mammals such as tiger”, “mammals including tiger”, “mammals like tiger”, “mammals especially tiger”, “tiger and/or other mammals” etc. These patterns are used for another web search and from the search result the list of final candidates(FC) are generated. After that the candidates in the list are verified. Finally the verified instances are added with the ontology classes to perform the annotation.

3.4 Verification of Candidate Instances

OntoStore makes web searches for instance extraction in two different phases. From the first search result it generates a list of initial candidate instances for an ontology class and these candidate instances are used for the second web search. The second web search is done in order to calculate the support value of an instance. The support(S) is calculated for each pair(i,c) by counting how many times the pair(i,c) has occurred in the result pages(r) for each pattern(p) :

$$S_{(i,c)} = \qquad /r$$

For example if the pair (*tiger, mammal*) occurs 10 times in 50 result pages the pair (*tiger, mammal*) has the support value 0.2.Candidates having support(S) more than a threshold value(T_H) are considered as final instances. The threshold value for a class is determined by analyzing the support value of all extracted instances of a class.

Chapter 4

Experimental Analysis

Here we have used Java, JSON, JENA Ontology API and Netbeans IDE for implementing the prototype of OntoStore. JSON was used for making web searches using the Google API. JENA Ontology API was used for accessing ontology classes. Here we have simulated the OntoStore prototype for the ontology described in figure-1.

4.1 Pattern Generation

The ontology classes were traversed in a reverse manner to generate patterns by adding Hearst Phrases. These patterns were submitted for making web searches to extract initial candidates. The initial candidates were added with the previous patterns to generate new patterns which were also submitted for another web search. The purpose of the second web search is to verify the initial candidates. The pattern generation process is described in details in table 1.

Table 1: Pattern generation process

Class Name	Hearst Phrase	Pattern	Instance list	New Pattern
Organism	such as including like especially and other or other	Organism such as Organism including Organism like Organism especially and other Organism or other Organism	amoeba paramecium bacterium virus …	Organism such as amoeba Organism including amoeba Organism like amoeba Organism especially amoeba amoeba and other Organism amoeba or other Organism
Plant	such as including like especially and other or other	Plant such as Plant including Plant like Plant especially and other Plant or other Plant	orne trematode zoonoses mascoma …	Plant such as orne Plant including orne Plant like orne Plant especially orne orne and other Plant orne or other Plant
Animal	such as including like especially and other or other	Animal such as Animal including Animal like Animal especially and other Animal or other Animal	human horse birds …	Animal such as human Animal including human Animal like human Animal especially human human and other Animal human or other Animal
Mammal	such as including like especially and other or other	Mammal such as Mammal including Mammal like Mammal especially and other Mammal or other Mammal	elephant dolphins whales …	Mammal such as elephant Mammal including elephant Mammal like elephant Mammal especially elephant elephant and other Mammal elephant or other Mammal
Fish	such as including like especially and other or other	Fish such as Fish including Fish like Fish especially and other Fish or other Fish	tuna sardine salmon …	Fish such as tuna Fish including tuna Fish like tuna Fish especially tuna tuna and other Fish tuna or other Fish

4.2 Instance Extraction

Here we extracted the instances of each ontology class. For each instance we calculated the support value by counting how many times the pattern has occurred within the Google abstract of the search result. The following tables contain a list of extracted instances along with their class and support value.

Table 2: List of extracted instances for the class Organism

Instance	Class	No. of pages	Total occurrence	Support
human	Organism	48	48	1.00
amoeba	Organism	48	15	0.3125
paramecium	Organism	48	17	0.3541
virus	Organism	48	32	0.67
yeast	Organism	48	38	0.79
fungus	Organism	48	26	0.54
bacteria	Organism	48	48	1.00
living	Organism	48	15	0.31
hydra	Organism	48	8	0.17
cell	Organism	48	29	0.604
protozoan	Organism	48	5	0.104
organism	Organism	48	8	0.17

Table 3: List of extracted instances for the class Mammal

Instance	Class	No. of pages	Total occurrence	Support
whale	Mammal	48	25	0.52
elephant	Mammal	48	25	0.52
dolphin	Mammal	48	22	0.458
seal	Mammal	48	17	0.354
mouse	Mammal	48	28	0.58
bat	Mammal	48	27	0.56
primate	Mammal	48	19	0.39
veggie	Mammal	48	1	0.02
meat	Mammal	48	2	0.04
mafist	Mammal	48	0	0.0
hippopotamus	Mammal	48	1	0.02
rhinoceros	Mammal	48	0	0.0

Table 4: List of extracted instances for the class Fish

Instance	Class	No. of pages	Total occurrence	Support
sardines	Fish	48	45	0.93
herring	Fish	48	47	0.97
anchovies	Fish	48	48	1.00
catfish	Fish	48	43	0.89
cichlids	Fish	48	44	0.91
tuna	Fish	48	42	0.875
mackerel	Fish	48	45	0.94
salmon	Fish	48	41	0.85
spinach	Fish	48	4	0.08
leafy	Fish	48	6	0.125
anglers	Fish	48	18	0.375
angelfish	Fish	48	42	0.875
sharks	Fish	48	42	0.875

4.3 Result Analysis

After extracting the candidate instances and calculating their support value we analyze the support value of the instances. Here we can see even some valid instances of a class have poor support value as well as some invalid instances have satisfactory support value. For this reason candidates that has support value greater than a certain value called threshold value(T_H) are considered as valid instances. To determine the value of T_H we analyze the data collected in table 1, 2 and 3.

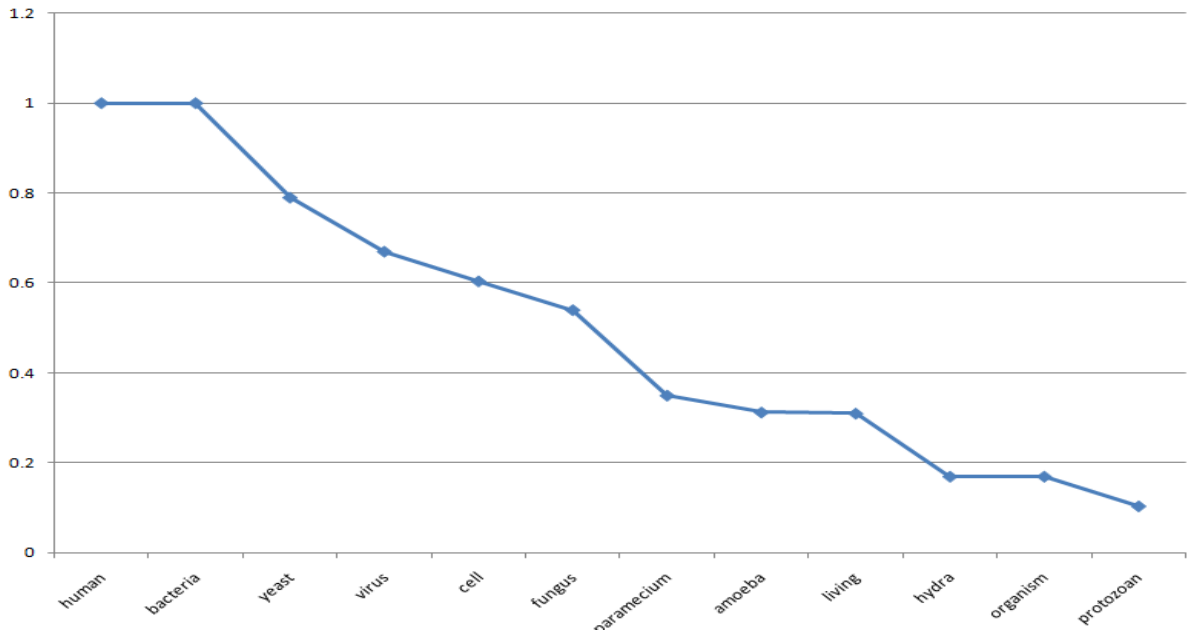

Figure-8: Support value for instances of class Organism

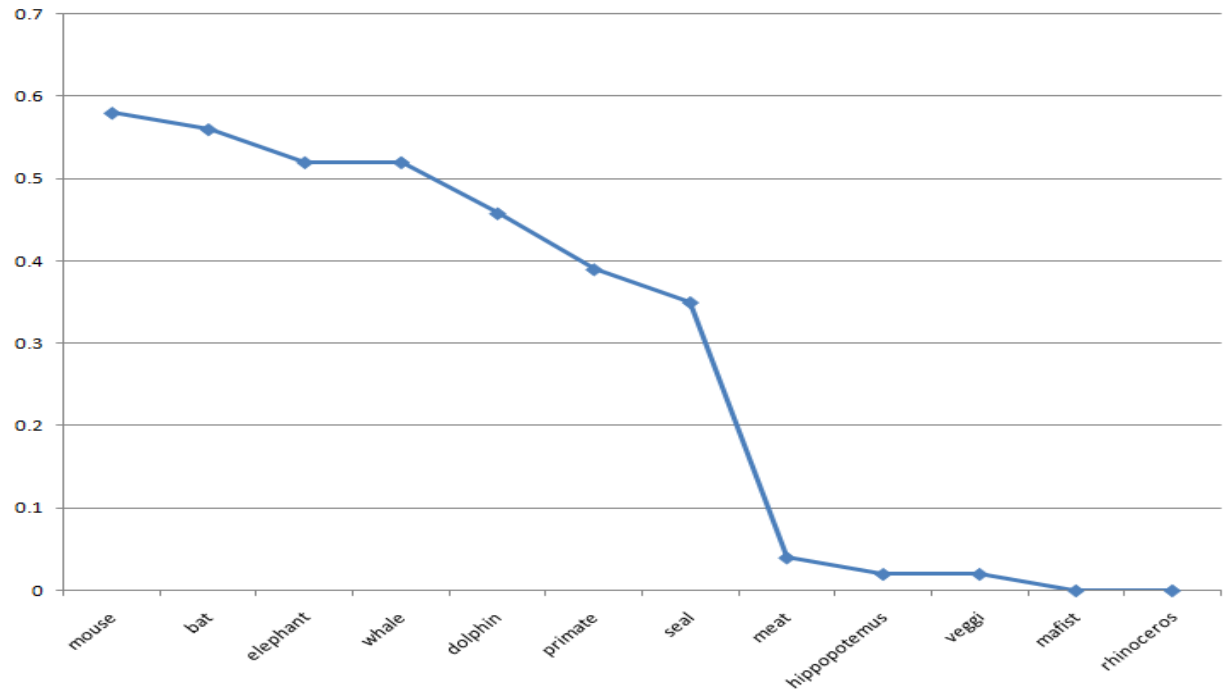

Figure-9: Support value for instances of class Mammal

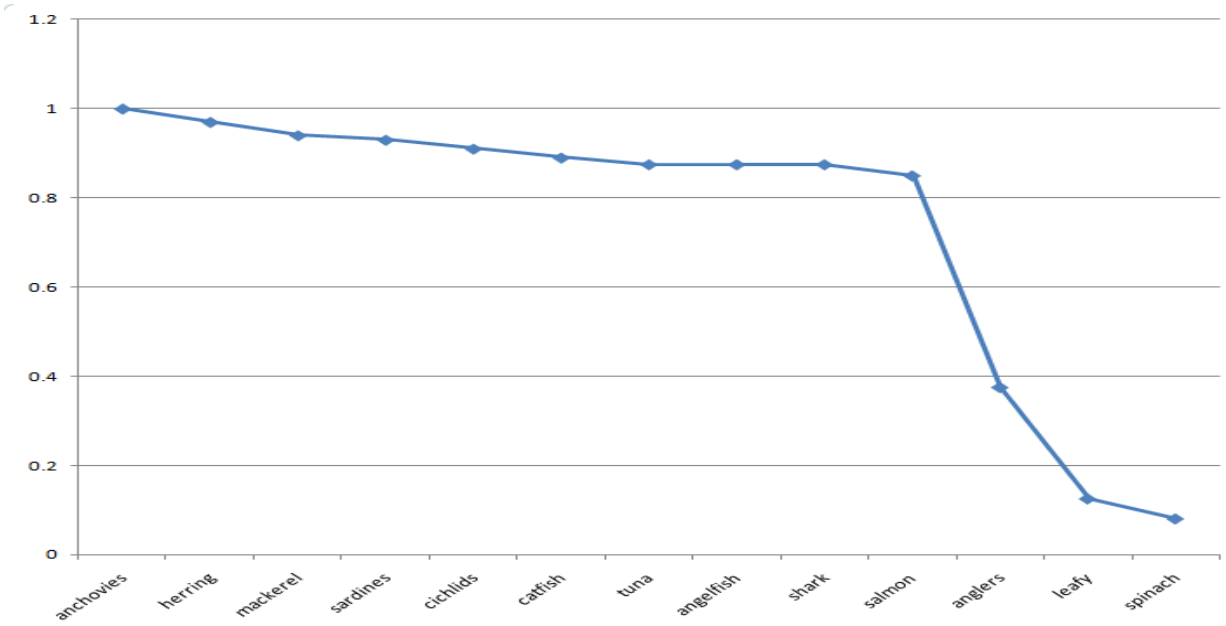

Figure-10: Support value for instances of class Fish

After analyzing the support values for the instances of different classes we consider that instances that have support value more than 0.5 can be considered as valid instances. Therefore ,

Threshold value (T_H) = 0.5

Chapter 5

Related Works

The operation of OntoStore is completely automated. Similar sort of approach is used by OntoSyphon[1] an ontology-driven information extraction system. It also operates in an automatic and unsupervised manner with less human intervention in order to learn all possible information about an ontology available in the web. It extracts relatively shallow information about the relations and instances of an ontology class from the web using Binding Engine[9]. PANKOW[3] is a document-driven domain independent system which annotates a given set of web pages by extracting the candidate proper nouns and finding the class of them. It uses the number of web pages in which a certain patterns appears to calculate the strength of it. C-PANKOW[2] which is also a document-driven domain independent system is the successor of PANKOW[3]. It scans through the web page for candidate instances and uses a pattern library to execute Google query. Rather than downloading the entire search result pages it uses the Google abstract. The operation of OntoSyphon[1] has similarity with the operation of OntoStore. OntoSyphon[1] traverse through the ontology classes in a top-down manner but OntoStore uses a bottom-up approach. OntoSyphon[1] uses the redundancy for instance verification but OntoStore uses the threshold support value(T_H) for the verification process. Both PANKOW[3] and C-PANKOW[2] annotates a given set of documents by finding the classes of the candidate proper nouns. All the candidates may not belong to the same ontology as a result by annotating the web page all the instances of an ontology may not be found. On the other hand, the main focus of OntoStore is to enrich ontologies by finding out all the instances of it. The process of OntoSyphon, Armadillo and PANKOW are described below in details.

5.1 OntoSyphon – an ontology driven domain-independent approach:

OntoSyphon is an alternative ontology-driven information extraction (IE) system. Instead of sequentially handling documents, OntoSyphon processes an ontology in some order. For each ontological class or property, OntoSyphon searches a large corpus for instances and relations than can be extracted. The redundancy in the web and information in the ontology is used to verify the candidate instances, subclasses, and relations that were found. Compared to more traditional document-driven IE, OntoSyphon's ontology-driven IE extracts relatively shallow information from a very large corpus of documents, instead of performing more exhaustive (and expensive) processing of a small set of documents. Instead of trying to learn all possible information about a particular document, it focuses on particular parts of an ontology and try to learn all possible information about those ontological concepts from the web.

OntoSyphon operates in a fully automatic, unsupervised manner, and uses the web rather than a domain-specific corpus be identified. The algorithm of OntoSyphon is stated below:

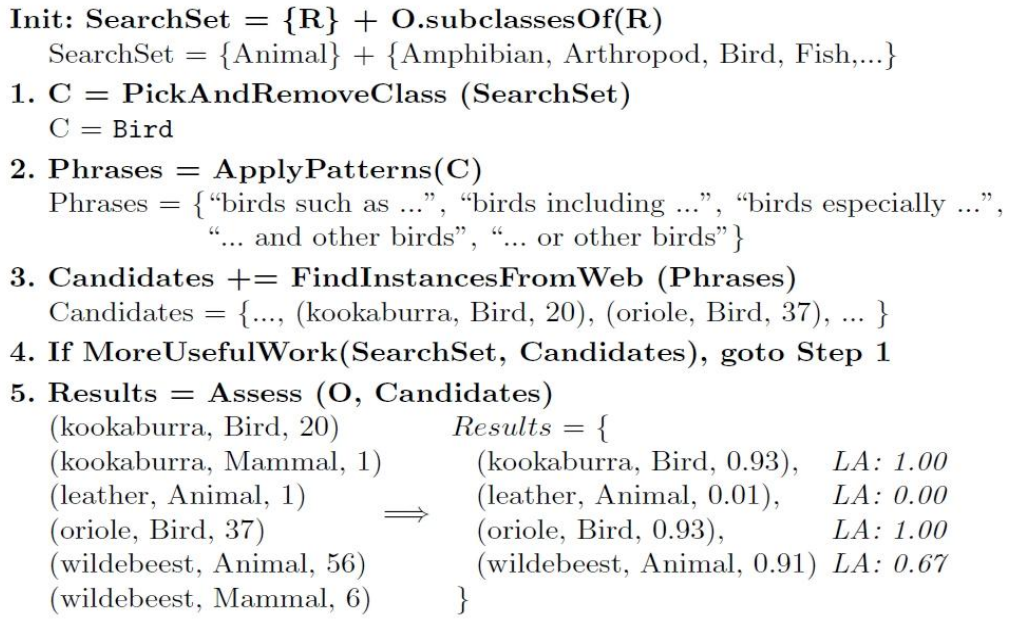

Figure-11: OntoSyphon's algorithm (bold lines), given a root class R, for populating an ontology O with instances, and partial sample output (other lines). The text (oriole, Bird, 37) describes a candidate instance that was extracted 37 times. Step 5 converts these counts into a confidence score or a probability, and chooses the most likely class for candidates that had more than one possible class. "LA" is the "Learning Accuracy" of the final pair.

5.1.1 Operation of OntoSyphon :

Figure 4 gives pseudocode for OntoSyphon's operation. The input to OntoSyphon is an ontology O and a root class R such as Animal. The search set is initialized to hold the root term R and all subclasses of R. OntoSyphon then performs the following steps: pick a "promising" class C from the ontology (step 1), instantiate several lexical phrases to extract instances of that class from the web (steps 2-3), then repeat until a termination condition is met (step 4). Finally, use the ontology and statistics obtained during the extraction to assess the probability of each candidate instance (step 5). Below it is explained in more detail.

1) **Identify a Promising Class**: OntoSyphon must decide where to focus its limited resources. For initial experiments, it was pragmatically chosen to completely explore all subclasses of the user-provided root class.
2) **Generate Phrases**: Given a class C, lexico-syntactic phrases are searched that indicate likely instances of C. For instance, phrases like "birds such as" are likely to be followed by instances of the class Bird. Five Hearst phrase templates were used in the sample output of Figure 4.
3) **Search and extract**: Next, it searches the web for occurrences of these phrases and extract candidate instances. This could be done by submitting the phrases as queries to a search engine, then downloading the result pages and performing extraction on them. For efficiency, the Binding Engine (BE) was used. BE accepts queries like "birds such as

<NounPhrase>" and returns all possible fillers for the <NounPhrase> term in about a minute, but for only a 90-million page fragment of the web.

4) **Repeat** : The entire process is repeated until the SearchSet is empty.
5) **Assess Candidate Instances** : The final step where the extracted instances are evaluated.

Overall, we can conclude that it is possible extracting instances from a web corpus using OntoSyphon. But it has some limitations. OntoSyphon is not suited for populating every kind of ontology. For instance, ontologies describing things or events that are mentioned only a handful of times on the web are not well suited to the current strategy of using simple pattern-based extractions followed by redundancy based assessment. Likewise, classes that are either complex (NonBlandFish) or ambiguous (Player) will not yield good results.

5.2 Domain-specific annotation with Armadillo:

Armadillo is a system for producing automatic domain-specific annotation on large repositories in a largely unsupervised way. It annotates by extracting information from different sources and integrating the retrieved knowledge into a repository. The repository can be used both to access the extracted information and to annotate the pages where the information was identified. Also the link with the pages can be used by a user to verify the correctness and the provenance of the information. Armadillo's approach is illustrated in Figure 4.

Input:

- an Ontology;
- an Initial Lexicon;
- a Repository of Documents;

Output: A set of triples representing the extracted information and to be used to annotate documents

do {

- spot information using the lexicon
- seek for confirmation of the identified information
- extend lexicon using adaptive information extraction and seek confirmation of the newly extracted information

} **while** a stable set of information is found(e.g. the base does not grow anymore)

- Integrate Information from different documents
- Store information in repository

Figure-12: Armadillo's main algorithm

In the first step in the loop, possible annotations from a document are identified using an existing lexicon (e.g. the one associated to the ontology). These are just potential annotations and must be confirmed using some strategies (e.g. disambiguation or multiple evidence). Then other annotations not provided by the lexicon are identified e.g. by learning from the context in which the known ones were identified. All new annotations must be confirmed and can be used to learn

some new ones as well. They will then become part of the lexicon. Finally all annotations are integrated (e.g. some entities are merged) and stored into a database.

Armadillo employs the following methodologies:

1) Adaptive Information Extraction from texts (IE): used for spotting information and to further learning new instances.
2) Information Integration (II): used to
   (i) discover an initial set of information to be used to seed learning for IE
   (ii) confirm the newly acquired(extracted) information, e.g. using multiple evidence from different sources. For example, a new piece of information is confirmed if it is found in different (linguistic or semantic) contexts.
3) Web Services: the architecture is based on the concept of ”services”. Each service is associated to some part of the ontology (e.g. a set of concepts and/or relations) and works in an independent way. Each service can use other services (including external ones) for performing some sub-tasks. For example a service for recognizing researchers names in a University Web Site will use a Named Entity Recognition system as a sub-service that will recognize potential names (i.e. generic people’s names) to be confirmed using some internal strategies as real researchers names (e.g. as opposed to secretaries’ names).

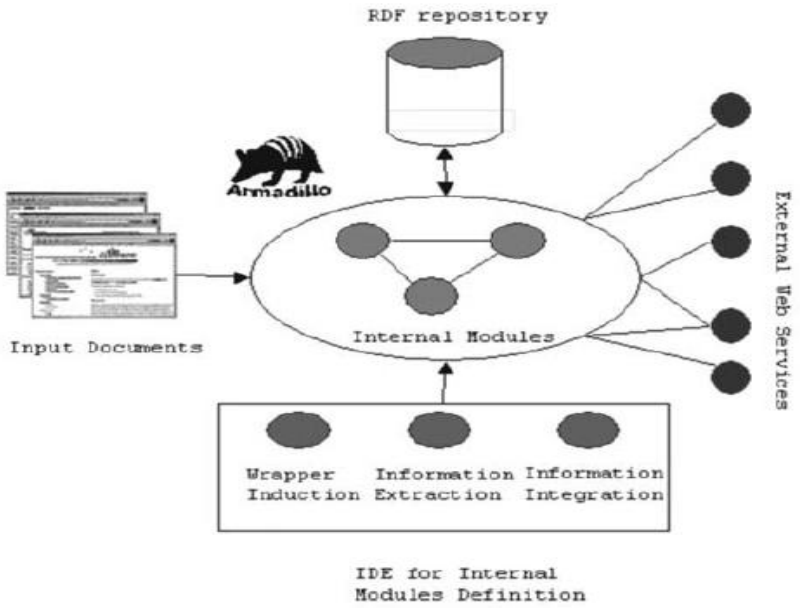

Figure-13: The Armadillo architecture

4) RDF repository: where the extracted information is stored and the link with the pages is maintained.

Overall we can say that, it is possible to annotate some web documents using Armadillo automatically. But All the annotations are not reliable. Many IE systems are able to learn from completely annotated documents only, so that all the annotated strings are considered positive examples and the rest of the text is used as a set of counterexamples. This means that the system

is presented with positive examples, but the rest of the texts can never be considered as a set of negative examples, because unannotated portions of text can contain instances that the system has to discover, not counterexamples.

5.3 PANKOW (Pattern-based Annotation through Knowledge on the Web):

5.3.1 The process of PANKOW:

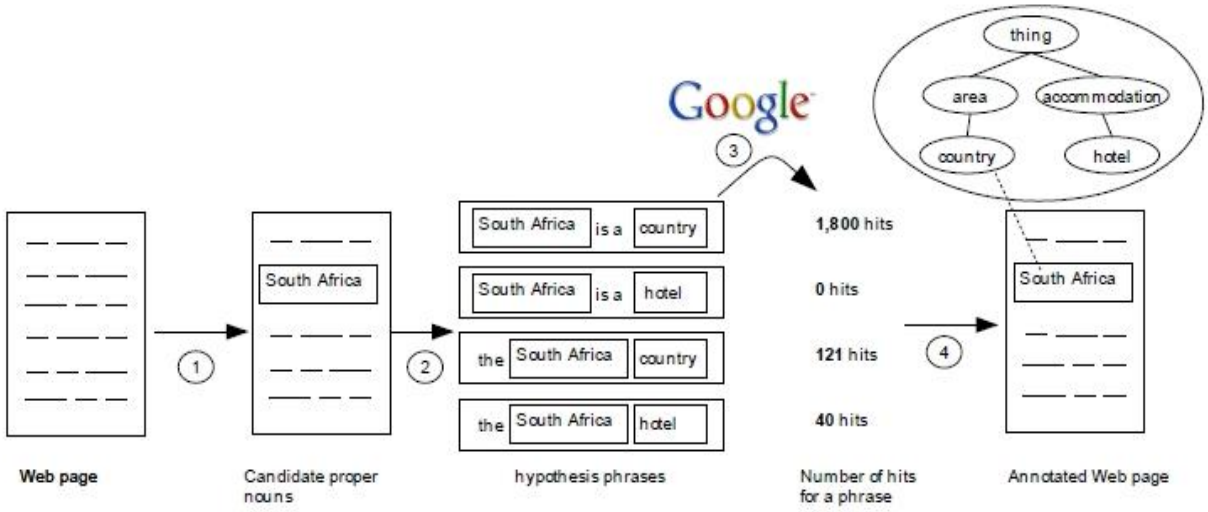

Figure-14: The process of PANKOW

Input: A web page.

Step 1: The system scans the Web page for phrases in the HTML text that might be categorized as instances of the ontology. Candidate phrases are proper nouns, such as '*Nelson Mandela*', '*South Africa*', or '*Victoria Falls*'). A parts-of- speech tagger (cf. Section 3 and Section 5) is used to find such candidate proper nouns. Thus, we end up with a
Result 1: Set of candidate proper nouns

Step 2: The system iterates through the candidate proper nouns. It introduces all candidate proper nouns and all candidate ontology concepts into linguistic patterns to derive hypothesis phrases. For instance, the candidate proper noun '*South Africa*' and the concepts Country and Hotel are composed into a pattern resulting in hypothesis phrases like '*South Africa is a country*' and '*South Africa is a hotel*'.
Result 2: Set of hypothesis phrases.

Step 3: Then, Google™ is queried for the hypothesis phrases through its Web service API (Section 3.2). The API delivers as its results
Result 3: The number of hits for each hypothesis phrase.

Step 4: The system sums up the query results to a total for each instance-concept pair. Then the system categorizes the candidate proper nouns into their highest ranked concepts (cf. Section 3.3). Hence, it annotates a piece of text as describing an instance of that concept. Thus we have
Result 4: An ontologically annotated web page.

Chapter 6

Conclusion

Even though OntoStore introduces a different approach in web annotation it still has to face some challenges. Among them time complexity is the most crucial one. OntoStore is supposed to perform many queries in the web using Google. So it must perform the searching operations within an acceptable time limit. For this reason we are using the Google abstract instead of downloading the entire page for instance extraction for every web search. In order to ensure correctness of the extracted instances we are using threshold support (T_H) value. In future we intend to determine the threshold value (T_H) automatically by adjacent class verification. For example for verifying the candidates of class *mammal* the adjacent class fish will be selected for generating patterns like “fish such as tiger” , “fish including tiger” etc. to perform another web search. From this search result we can get the support value(S) of the instances for the class *fish*. We can compare these two support values and determine the threshold value (T_H) automatically. But the adjacent class verification process also requires a large number of queries from Google which can be time consuming as well. To handle these issues we ensured the depth of searching using OntoStore can be changed if required. We hope that our future works will increase the efficiency and reliability of OntoStore.

References

[1] McDowell, L., Cafarella, M.: Ontology-driven Information Extraction with OntoSyphon. In: Fifth int. Semantic Web Conference. (ISWC 2006)
[2] Cimiano, P., Ladwig, G., Staab, S.: Gimme' the context: Context-driven automatic semantic annotation with C-PANKOW. In: Proc. of the Fourteenth Int.WWW Conference. (2005)
[3] Cimiano, P., Handschuh, S., Staab, S.: Towards the Self-Annotating Web. In: Proc. of the Thirteenth int. WWW Conference. (2004)
[4] Champa, S.,Dingli, A., Ciravegna, F.: Armadillo: harvesting information for the semantic web. In: Proc. of the 27th Annual Int. ACM SIGIR conference on Research and development in information retrieval. (2004)
[5] Etzioni O., Fader A., Christensen J., Soderland S., Mausam : Open Information Extraction: the second generation. In: International Joint Conference on Artificial Intelligence. (2011)
[6] Fader A., Soderland S., Etzioni O.: Identifying relations for open information extraction. In: Conference on Empirical Methods in Natural Language Processing (2011)
[7] Guha, R., McCool, R., Miller, E.: Semantic search. In: World Wide Web. (2003)
[8] Celjusca, D., Vergas-Vera, M.: OntoSophie: A semi-automatic system for ontology population from text. In: International Conference on Natural Language Processing (ICON). (2004)
[9] Fader A., Soderland S., Etzioni O.: Extracting Sequence from the web. In: Annual Meeting of the Association for Computational Linguistics. (2010)
[10] Etzioni O., Soderland S., Weld D., Banko M.: Open information extraction from the web. In: Communications of the ACM. (2008)
[11] Davalcu, H., Vadrevu, S., Nagarajan, S.: OntoMiner: Bootstrapping and populating ontologies from domain specific web sites. IEEE Intelligent Systems 18(5) (2003) 24-33
[12] Dill, S., Eiron, N., Gibson, D., Gruhl,D., Guha, R.: Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In: Proc. of the Twelfth Int. WWW Conference. (2003)
[13] Cafarella, M., Etzioni, O.: A search engine for natural language applications. In: Proc. of the Fourteenth Int.WWW Conference. (2005)
[14] Hearst, M.: Automatic acquisition of hyponyms from large text corpora. In: Proc. of the Fourteenth Int.Conf.on Computational Linguistics. (1992)