

12-2009

# Semantic Search

Anup Sawant

Clemson University, [asawant@clemson.edu](mailto:asawant@clemson.edu)

Follow this and additional works at: [https://tigerprints.clemson.edu/all\\_theses](https://tigerprints.clemson.edu/all_theses)

 Part of the [Computer Sciences Commons](#)

---

## Recommended Citation

Sawant, Anup, "Semantic Search" (2009). *All Theses*. 744.

[https://tigerprints.clemson.edu/all\\_theses/744](https://tigerprints.clemson.edu/all_theses/744)

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact [kokeefe@clemson.edu](mailto:kokeefe@clemson.edu).

# SEMANTIC WEB SEARCH

---

A Thesis  
Presented to  
the Graduate School of  
Clemson University

---

In Partial Fulfillment  
of the Requirements for the Degree  
Master of Science  
Electrical Engineering

---

by  
Anup Satish Sawant  
December 2009

---

Accepted by:  
Dr. Robert Schalkoff, Committee Chair  
Dr. John Gowdy  
Dr. Ian Walker

# Abstract

The thesis describes a Semantic approach towards web search through a stand-alone Java application. An Ontology Web Language(OWL) model is used to build a knowledge database related to different types of Organisms. The goal is to guide the Google web search engine using this OWL model. In the first approach towards Semantic web search, an inference engine called CLIPS is used and in the second approach, the Protege-OWL API is used. The thesis goes in detail about the design, working and comparison of these two approaches. The thesis also deals with design approach for enhancement of the OWL model, once the Semantic web search is done through the Protege-OWL API. This is achieved using Natural Language Processing and Parsing technique. Examples and results of the search and enhancement part of the application are described in detail. Future research directions are indicated.

# Dedication

I dedicate this thesis to my mother, father and sister who have always been supportive throughout my career.

# Acknowledgments

I would like to thank Dr. Schalkoff for providing me with the opportunity and the resources to pursue this thesis topic. I also would like to thank Dr. Schalkoff, Dr. Gowdy and Dr. Walker, for their technical guidance during my course of study at Clemson.

# Table of Contents

<b>Title Page</b> . . . . .	<b>i</b>
<b>Abstract</b> . . . . .	<b>ii</b>
<b>Dedication</b> . . . . .	<b>iii</b>
<b>Acknowledgments</b> . . . . .	<b>iv</b>
<b>List of Figures</b> . . . . .	<b>vi</b>
<b>1 Introduction</b> . . . . .	<b>1</b>
1.1 Motivation . . . . .	1
1.2 The Objective . . . . .	3
1.3 Overview of the Thesis . . . . .	4
1.4 OWL vs Frames . . . . .	4
1.5 Brief description of the software . . . . .	5
<b>2 Design of Semantic web search using the CLIPS, CLIPSOWL and the Protege-OWL editor</b> . . . . .	<b>8</b>
<b>3 Design of Semantic web search using the Protege-OWL API</b> . . . . .	<b>11</b>
3.1 Query-Ontology interaction and its impact on search . . . . .	13
3.2 Advantages over CLIPS . . . . .	15
<b>4 Ontology improvement with NLP search and Parsing technique</b> . . . . .	<b>17</b>
<b>5 Examples</b> . . . . .	<b>22</b>
5.1 Working of application with examples . . . . .	22
5.2 Code description . . . . .	37
<b>6 Conclusions and Discussion</b> . . . . .	<b>38</b>
6.1 Semantic Search . . . . .	38
6.2 Further scope of study . . . . .	39
<b>Bibliography and Software References</b> . . . . .	<b>42</b>

# List of Figures

1.1	Do you see the difference ? . . . . .	2
2.1	Semantic application using CLIPS. . . . .	9
3.1	Semantic application using Protege API. . . . .	12
4.1	Ontology improvement using NLP and Parsing technique. . . . .	18
5.1	Ontology used for Semantic Search . . . . .	23
5.2	Striped Individual before search . . . . .	24
5.3	Search result for Striped Dolphin . . . . .	25
5.4	Google search result for keyword 'Striped' . . . . .	27
5.5	Properties added to Striped Individual . . . . .	28
5.6	Properties shown on webpage . . . . .	29
5.7	Results after selection of 'have[melon]' property . . . . .	31
5.8	Addition of new properties . . . . .	32
5.9	Search results for query 'Cricket' . . . . .	33
5.10	New properties added to Cricket Individual . . . . .	34
5.11	Search results for query 'Latex' . . . . .	35
5.12	New properties added to Latex Individual . . . . .	36

# Chapter 1

## Introduction

### 1.1 Motivation

”There is nothing like looking, if you want to find something. You certainly usually find something, if you look, but it is not always quite the something you were after.”– J.R.R. Tolkien

With growth of the internet over the past decades search engine has become an important entity in everyday life. Search engines mainly crawl the web and create huge repository of webpages. Given a query, search for a target webpage is based on text matching and popularity based ranking many a times. Although, the results are good enough, it is seen that not all the search results are relevant to user’s query. Depending on the context, a word can have several meanings and hence text based search produces all possible results of webpages in which the given word from user query appears with high frequency. In this case, the results need not be accurate and according to user’s needs.

Problems of inaccurate results could be solved if we somehow try to decipher the meaning of the user query or enhance the user query. Hence, semantic knowledge



The image shows a side-by-side comparison of search results for the term "M4".

**Left Panel (Semantic Search):**

- Browser address bar: `http://localhost:8181/GoProSeman`
- Page title: **Semantic Search**
- Search input: "M4" with a "Search" button.
- Results:
  - [CARMODEL - M4 - 1/43 1/24 1/18 - model ca](#)  
Zoom ALFA ROMEO - 33.3 TT N 19 24h LE I  
1/43 ...  
[www.carmodel.net](http://www.carmodel.net)
  - [M4 - Automodelli per collezionisti - Scale mo](#)  
M4 produces **car models** of years 40/50/60  
and thanks to ...  
[www.m4modelcars.it](http://www.m4modelcars.it)
  - [Toy Car Model :B2BManufactures.Com For](#)  
Jia Dyi Enterprise Co., Ltd. : Toy **Car Model**  
[www.manufacturers.com.tw](http://www.manufacturers.com.tw)
  - [YouTube - High-Poly Car Model Animation](#)  
My first high-poly **car model** animation.For r  
[www.youtube.com](http://www.youtube.com)
- Navigation: 1 2 3 4 5 6 7 8 [More results >](#)

**Right Panel (Google Search):**

- Browser address bar: `http://localhost:8181/GoProSeman`
- Page title: **Google**
- Search input: "M4"
- Buttons: [Web](#) [Show options...](#) [Compare on eBay](#)
- Section: **Image results for M4** - [Report images](#)
- Images:
  - Top left: "M4 CARBINE" parts kit.
  - Top right: M4 carbine rifle.
  - Bottom left: M4 carbine rifle.
  - Bottom right: M4 carbine rifle.
- Text results:
  - [M4 carbine - Wikipedia, the free encycloped](#)  
The **M4** carbine is a family of firearms tracing its line:  
M16, all based on the original AR-15 made by ArmaL  
[Overview](#) - [History and variants](#) - [Design](#) - [Effectivene:](#)  
[en.wikipedia.org/wiki/M4\\_carbine](http://en.wikipedia.org/wiki/M4_carbine) - [Cached](#) - [Similar](#)
  - [m4 \(computer language\) - Wikipedia, t](#)  
Sep 15, 2009 ... **m4** is a general purpose mac  
Dennis Ritchie. The name "**m4**" stands for "m:  
[en.wikipedia.org/wiki/M4\\_\(computer\\_language\)](http://en.wikipedia.org/wiki/M4_(computer_language))

Figure 1.1: Do you see the difference ?

plays a vital role. Semantics is the study of meaning. Information could be stored in Ontology form and closest meaning of user query can be derived. User query can be enhanced and expanded to retrieve better and focused results. Hence, an attempt has been made to do Semantic web search using the Google API and a local Ontology. The above figure shows the difference when user puts M4 as a query in a search engine driven by semantics vis-a-vis a search engine like Google that is based on text based popularity ranking. Given a knowledge domain of Car a Semantic engine would derive search results related to M4 Car models and not M4 weapons which are certainly more popular for Google like search engine to bring them up first on result page.

## 1.2 The Objective

The objective of this research is to implement a focused web search in the context of an Ontology. A normal text based search can be driven on the basis of knowledge present in the form of an Ontology. The thesis aims to use this Ontology and develop a stand-alone application to fetch more accurate and relevant search results for a query vis-a-vis search engines based on text matching. The thesis also aims to leverage the Ontology by extracting more information from web pages through Natural Language Processing technique and update or enhance the ontology.

This work is comprised of following three parts:

- (1) Semantic web search using the CLIPS as an inference engine.
- (2) Semantic web search using the Protege-OWL API.
- (3) Ontology improvement following query results with Natural Language Processing and Parsing technique.

## 1.3 Overview of the Thesis

The thesis is organized as follows

Chapter 2 describes design of the Semantic application using the CLIPS, CLIP-SOWL and the Protege-OWL editor.

Chapter 3 describes design of the Semantic application using the Stanford Protege-OWL API and the Google API.

Chapter 4 suggests a method of improvement or enhancement of the local Ontology and design of whole Semantic application using the Stanford Lex-Parser and the HTML parser.

Chapter 5 talks about related work with examples.

Chapter 6 concludes the thesis summarizing the developed Semantic web search application and further scope of improvement.

## 1.4 OWL vs Frames

Knowledge modeling can be done in many ways. In Protege, we may use either Frame based or OWL based representations. Frames have been used to develop tools and ontologies for specific user communities. They are not compatible with World Wide Web. Frames are based on Unique Name Assumption(UNA) i.e. different names refer to different things, by default. OWL(Ontology Web Language) models on the other hand are compatible with World Wide Web and add certain capabilities to ontologies like, distribution across many systems and scalability to Web needs. This is the main reason why we use an OWL model in order to develop Semantic web application. Furthermore, OWL adds more vocabulary to describe properties, relation between classes, cardinality, characteristics of properties etc., which Frames

don't support. In OWL models, different names can refer to the same thing. A programmer would want to be aware of the fact that in Frames, relations are known as Slots and the constraints on Slots are known as Facets, whereas in an OWL model, relations are known as Properties and the constraints on Properties are known as Restrictions.

## 1.5 Brief description of the software

1. Protege-OWL Editor and API: Protege-OWL[11] is an extension of Protege which supports Ontology Web Language(OWL). Protege-OWL also comes as an open source Java API called as Protege-OWL API[10]. It enables user to load, save, edit and visualize Ontologies. Protege-OWL is also integrated with Jena which is another open source Java API for querying and parsing OWL model.
2. Stanford NLP Lex-Parser: Stanford Lex-Parser[13] is the heart of the Semantic search. It works on grammatical structure of sentences. It is a java package and carries its own dictionary of words. Lex-Parser produces grammatical dependencies in various forms. Our Semantic search uses the typed dependency representation of Lex-Parser. Lex-Parser can process about 40-50 words at a time and derive dependencies among all the words. Typed dependencies are binary grammatical relations. The current representation of Lexparser contains 55 grammatical relations. The hierarchy of dependencies helps us to process and derive information in sequential manner.
3. CLIPS: CLIPS[15] is an expert system tool. It is written in C and provides support for three different programming paradigms: rule-based, object-oriented and procedural. CLIPS can be installed on different operating systems like

Windows, Mac and Unix without code changes. It can also be integrated with languages such as C, Java and FORTRAN. CLIPS provides CLIPSJNI, a Java Native for CLIPS. It helps a programmer to interface CLIPS engine with any Java application.

4. HTML Parser[14]: It is an open source Java library used to parse HTML pages. It can be used to extract text, extract link, ensure valid links, move existing web pages to XML etc. In Semantic Web Search, it removes HTML tags from a web page and returns the text in the form of string to the Stanford Lex-Parser for further manipulation.
5. Google API: Google AJAX search API[19] allows programmer to put Google search in web pages with JavaScript. A programmer can choose web search, image search, multimedia search or map search to embed in his own web pages. Semantic Web Search uses Google AJAX Web Search API to embed Google Search in web pages. It is a Javascript library that provides web objects which carry web search results. Google provides online code playground, example codes and tutorials to get comfortable with the Web Search API.
6. Brief overview of Application code: Semantic Web Search Application applies Model, View and Controller(MVC) pattern of coding using Servlet Technology. The Model part of the application comprises of classes which provide expanded query to Google API, parse HTML pages, find grammatical relations between words, extract information, update Ontology and verify user query. The View consists of Google API for retrieving web search results, Java Scriptlet for dynamic user options, front end and interaction with Java Servlets. Controller consists of Java Servlets to provide communication between View and Model

part of the application. These parts will be discussed in detail in further chapters.

## Chapter 2

# Design of Semantic web search using the CLIPS, CLIPSOWL and the Protege-OWL editor

C Language Integrated Production System (CLIPS)[15] is a widely known and free to use tool for building expert systems. It incorporates object oriented language COOL to write expert systems. CLIPS can read a .clp file and load classes, functions and instances to form a database which we can query. The Protege Frame based Ontology editor[12] uses the CLIPS text file format as its default save/load file format for both classes and instances. It provides compatible CLIPS extensions like 'allowed-classes'facet,'slot-documentation'facet, and project inclusion. Also, the Protege files can be read directly into the CLIPS. However, for the Semantic Web Search Application with the CLIPS, we make use of the Protege-OWL editor[11] to form an OWL model. This owl model is stored as .owl file which is not compatible with the CLIPS. The solution is to either have an inference engine which could read an OWL model from .owl file or convert .owl to .clp file which is readable by the CLIPS.

Hence, in order to use the CLIPS we need a tool called CLIPSOWL[16] which satisfies the requirement of converting .owl to .clp file. The CLIPSOWL is based on the Pellet DL reasoner[17] and the OWLAPI[18] in order to map an OWL model to COOL syntax.

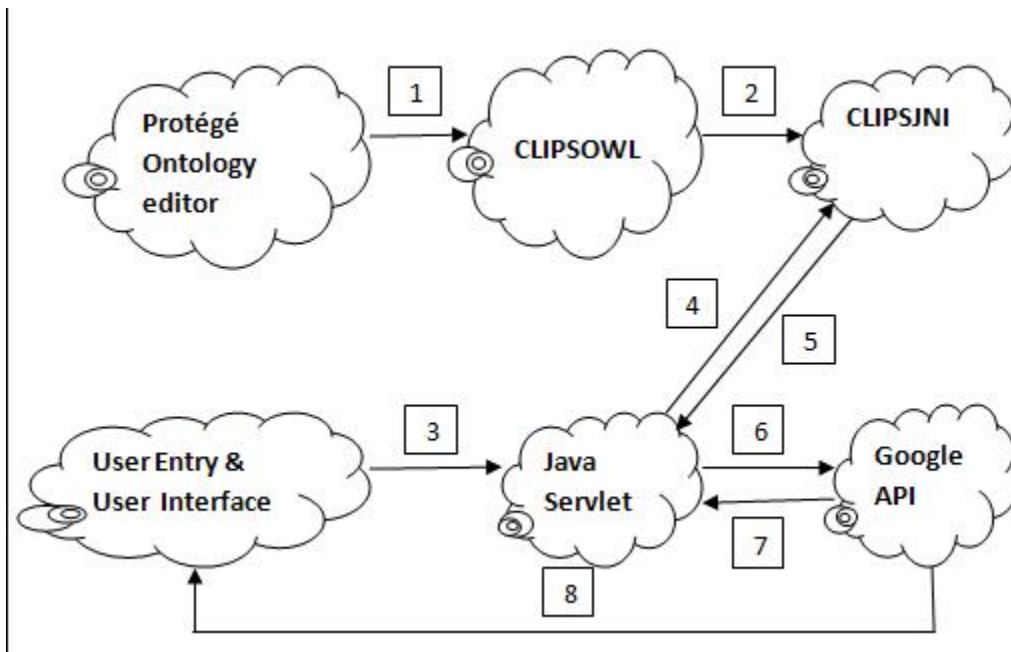


Figure 2.1: Semantic application using CLIPS.

The whole process of Semantic Search with the CLIPS can be explained as follows:

1. The Protege-OWL editor is used to create and edit an OWL model and store it in .owl file.
2. The programmer has to select the .owl file using the CLIPSOWL and convert it into the CLIPS readable .clp file. In order to expand user query, Model part of Semantic Search engine needs to interact with the CLIPS. This interaction with the CLIPS is provided by the CLIPSJNI. The CLIPS provides CLIPSJNI, a Java Native package. It helps a programmer to interface the CLIPS engine



with any Java application. The CLIPSJNI loads the .clp file and keeps the application ready for expansion of user query.

3. User needs to type a search query. The View part of the application sends user query to the Controller where query check and validation is done before it is sent for further enhancement.
4. After sufficient checks are done, user query is sent to the Model part of the application where the CLIPSJNI gets the enhanced query through COOL functions. For instance, if 'Hound' is the user query, the CLIPS COOL functions would help expand it to 'Hound is-a Dog is-a Mammal'.
5. The expanded and enhanced version of query is returned to the Controller.
6. The Controller passes on the modified query to the View which uses the Google API to make a web search. Note: A different design approach would be to use Java Applets instead of Servlets. During the course of application development, MVC Servlet Technology was found better than Java Applets for couple of reasons mentioned below:
  - (a) It takes considerable amount of time to load Applets in a browser.
  - (b) Applets can't load external libraries.
  - (c) Applets can't call native methods. This makes it difficult to run Semantic Search using the CLIPSJNI.
7. The Google API returns web search result object with webpage URLs to the View part of application.
8. User gets to see web search results based on knowledge derived from Ontology.

## Chapter 3

# Design of Semantic web search using the Protege-OWL API

An alternative approach towards developing a Semantic Web application can be using the Protege-OWL API[10] alone with the Google Web Search API[19]. The Protege-OWL API is centered around a collection of Java interfaces from the model package. These Java interfaces provide access to the OWL model and its elements like classes, properties and individuals. A programmer needs to operate on these interfaces in order to edit, update, make or save an OWL model. Given an OWL model, a programmer can query the model using methods like `OWLModel.getOWLNamedClass()`, `OWLModel.getOWLObjectProperty()` etc. which are a part of `OWLModel` class. The `OWLModel` class provides access to all the resources in the model. The Protege-OWL API supports Model-view-controller architecture. This means that the model stores the representation of the ontology data, and changes in that model trigger events which external representation can react to. It also helps in decoupling the entire process of event handling and enhancement of model. Protege-OWL API allows us to load an existing OWL file from a URL and save it on disk. In summary, the

Protege-OWL API has been designed for the development of components that are executed inside of the Protege-OWL editor's user interface and for the development of stand-alone applications.

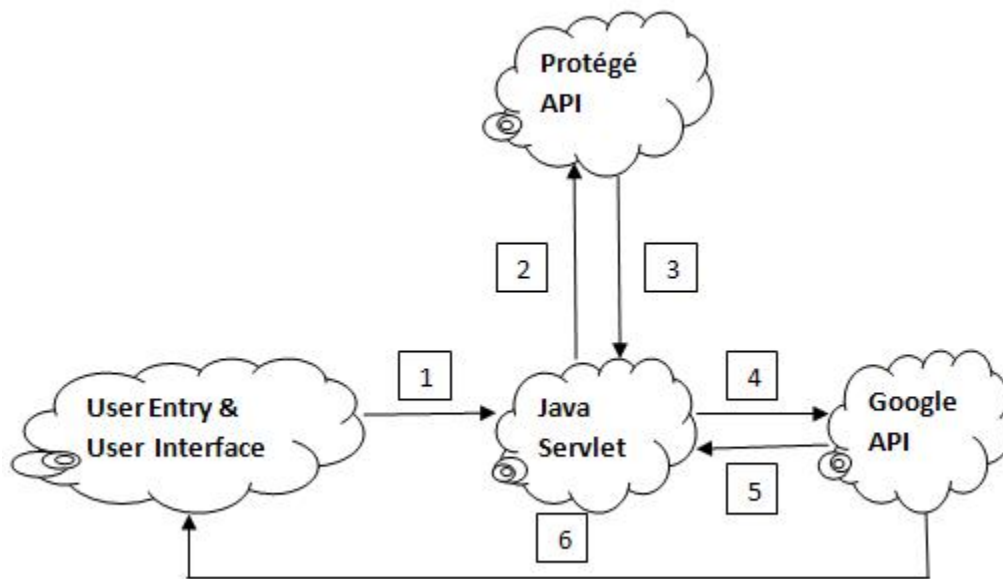


Figure 3.1: Semantic application using Protege API.

The whole process of Semantic Search with Protege-OWL API can be explained as follows:

1. The user needs to type a search query. The view part of the application sends user query to the Controller where query check and validation is done using the model class OWLhandler. OWLhandler has a method named `querycheck(String)` which helps Controller decide whether the user query is related to knowledge present in OWLModel and can be used for further processing. The Protege-OWL API interface methods like `getInstances()` and `getBrowserText()` help a programmer to travel through the OWLModel.

2. If the query is found acceptable then it is further sent for its enhancement by the Controller to the Model class OWLhandler. OWLhandler has a method named `getexpandedquery(String)` which helps retrieve information from the OWLModel. For instance, if 'Hound' is the user query, the Protege-OWL API and the Model class OWLhandler methods would help expand it to 'Hound is-a Dog is-a Mammal'.
3. The expanded and enhanced version of query is returned to the Controller.
4. The Controller passes on the modified query to the View which uses the Google API to make a web search.
5. The Google API returns web search result object with webpage URLs to the View part of application.
6. User gets to see web search results based on knowledge derived from Ontology.

### **3.1 Query-Ontology interaction and its impact on search**

For a given individual if the query is being hit for the first time then only the superclass details of individual are extracted from the Ontology and the query is enhanced. We can say only upper hierarchical information of an individual from an Ontology is used. On the other hand, if the query for a particular individual is being hit on selection of a property option then our application attempts to enhance the query using its superclass and selected property details. Following code snippet shows the process of query enhancement:

```

public String getexpandedquery(String query){
    try {
owlModel = ProtegeOWL.createJenaOWLModelFromURI(uri);
    } catch (OntologyLoadException e) {
e.printStackTrace();
    }

    Collection classes = owlModel.getUserDefinedOWLNamedClasses();

    /* Extract class information */
    for(Iterator it = classes.iterator(); it.hasNext());{
OWLNamedClass cls = (OWLNamedClass)it.next();
Collection instances = cls.getInstances(false);

    /* Extract individual */
    for(Iterator jt = instances.iterator(); jt.hasNext());{
OWLIndividual individual = (OWLIndividual)jt.next();
if(query.equalsIgnoreCase(individual.getBrowserText())){
    query += "+";
query += cls.getBrowserText();
    }
    }
    }
return query;
    }

    /***** javascript handles extraction of property values. *****/ /
    /***** Following is the code where property details are *****/ /
    /***** attached if user selects a property option *****/ /

```

```

        OWLhandler handle = new OWLhandler();
try {
if(append==null){
getanswer = handle.getexpandedquery(query);
        }
else{
getanswer = handle.getexpandedquery(query)+" "+append;
        }
System.out.println("query is :"+ getanswer);
} catch (Exception e) {
e.printStackTrace();
}

```

There is a trade-off between query enhancement and accuracy of results. If the property has multiple values then there is a chance of getting a defocused search. It is also true in the case where property has one value but parent class information is more. Enhanced query comes with more description and hence more words which ultimately degrade the performance of Google API search which determines best search results based on minimum query words, subject popularity, page rank and popularity of a page. A solution to this problem has been suggested in the Further Scope section of the last chapter.

## 3.2 Advantages over CLIPS

1. The CLIPS is written in C and hence compatibility issues need to be handled using the Java Native package CLIPSJNI in stand-alone Java applications like Semantic web search. The Protege-OWL API on the other hand is written in

Java and supports MVC architecture. Hence, it is easy to interface the Protege-OWL API with stand-alone web applications.

2. The CLIPS implements COOL and reads .clp files. The Protege-OWL editor makes and saves OWL model in .owl files and hence those files have to get converted to .clp using the CLIPSOWL (as described in Chapter 2) to reason the domain ontology by the CLIPS. The Protege-OWL API works directly with .owl file. A programmer can query the OWL model using API interface methods. Hence, there is no need to convert a .owl file to .clp file and vice versa.
3. Conversion from .owl to .clp and vice versa had to be done by the programmer. The Protege-OWL API helps to keep the whole process autonomous and hence also saves conversion time.

## Chapter 4

# Ontology improvement with NLP search and Parsing technique

Important domain knowledge details can be derived from result webpages and OWL model can be enhanced. This leads us to update the design described in previous chapters. In this chapter we try to discuss design method and issues of a stand-alone Semantic Web Search application where the underlying OWL model is enhanced autonomously to direct the user towards doing a focused search. The Stanford Lex-Parser[13] helps us to deduce grammatical relations between words in text. It is a java package which works on grammatical structure of sentences. The parser can read various forms of plain text input and can output various analysis formats, including part-of-speech tagged text, phrase structure trees, and a grammatical relations(typed dependency) format. In this application we use typed dependency representation of the Lex-Parser. It can process about 40-50 words at a time and derive dependencies among all the words. Typed dependencies are binary grammatical relations. The current representation of the Lexparser contains 55 grammatical relations.

The whole process of Semantic Search and Ontology improvement can be ex-



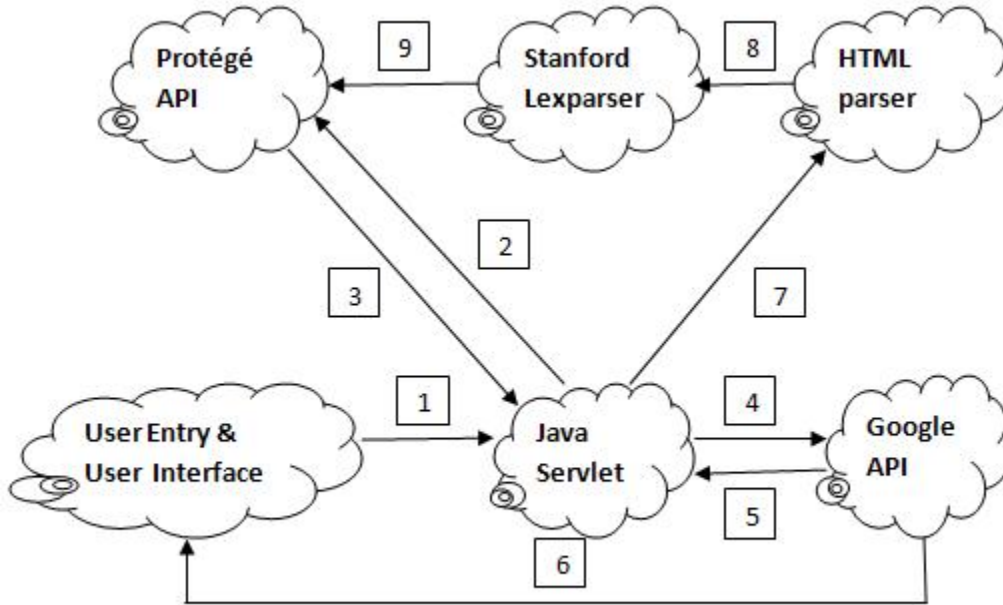


Figure 4.1: Ontology improvement using NLP and Parsing technique.

plained as follows:

1. User needs to type a search query. The view part of the application sends user query to the Controller where query check and validation is done using the model class OWLhandler. OWLhandler has a method named querycheck(String) which helps Controller decide whether the user query is related to knowledge present in OWLModel and can be used for further processing. The Protege-OWL API interface methods like getInstances() and getBrowserText() help a programmer to travel through the OWLModel.
2. If the query is found then it is further sent for its enhancement by the Controller to the Model class OWLhandler. OWLhandler has a method named getexpandedquery(String) which helps retrieve information from the OWLModel. For instance, if 'Hound' is the user query, the Protege-OWL API and the Model class

- OWLhandler methods would help expand it to 'Hound is-a Dog is-a Mammal'.
3. The expanded and enhanced version of query is returned to the Controller.
  4. The Controller passes on the modified query to the View which uses the Google API to make a web search.
  5. The Google API returns web search result object with webpage URLs to the View part of application.
  6. User gets to see web search results based on knowledge derived from Ontology.
  7. The Controller receives webpage URL from the Google API web search object and passes it on to the Model. The Model class ParseUrl uses the HTML parser[14] API to extract text from the web pages. The HTML parser can be used to extract text, extract link, ensure valid links, move existing web pages to XML etc. We use it to remove HTML tags from a web page and return the text content in the form of string. This is done with the help of StringExtractor(String) interface method under parserapplications package from the HTML parser API and parse(String,String) method of the Model class ParseUrl. Each web page is read once to determine the frequency of relevant query words present in it. Web pages are then ranked such that the top web page has highest frequency of query words present in it.
  8. The HTML parser passes first ranked web page to the Model class Lexparser which imports parser package and nlp.trees package from the Stanford Lex-Parser API. The Model class Lexparser has extractDetails(String,String,String) method which makes use of the Stanford Lex-Parser API englishPCFG.ser.gz dictionary to parse each sentence from the text sent by HTML parser. The

Stanford Lex-Parser API returns a tree structure of grammatical relations between words present in a sentence. For instance, if a given sentence is, 'Bell, based in Los Angeles, makes and distributes electronic, computer and building products.' then the relevant Stanford Typed Dependencies returned will be:

- (a) nsubj(makes-8, Bell-1)
- (b) nsubj(distributes-10, Bell-1)
- (c) partmod(Bell-1, based-3)
- (d) nn(Angeles-6, Los-5)
- (e) prep in(based-3, Angeles-6)
- (f) conj and(makes-8, distributes-10)
- (g) amod(products-16, electronic-11)
- (h) conj and(electronic-11, computer-13)
- (i) amod(products-16, computer-13)
- (j) conj and(electronic-11, building-15)
- (k) amod(products-16, building-15)
- (l) dobj(makes-8, products-16)
- (m) dobj(distributes-10, products-16)

This grammatical tree is read by the 'extractDetails' method and necessary information regarding query (which is treated as subject) is found. The Stanford Lex-Parser provides us with hierarchy of typed dependencies which makes it easy to deduce the relation relevant to particular subject.

9. After deriving necessary relations and extracting information, the Model class Lexparser updates OWL model through `updateModel(String,String,String)` method

of the Model class OWLhandler. The Protege-OWL API provides methods like addPropertyValue and setPropertyValue by which a programmer can update an OWL model on which he/she is working.

# Chapter 5

## Examples

### 5.1 Working of application with examples

The three parts discussed in last chapter were tried on Organisms Ontology. The goal is to search more information related to Plant and Animal kingdom organisms and improve on original Ontology. In this chapter we will discuss working of Semantic search application with examples. For instance, lets assume a particular user wants to search about Striped Dolphin. We will see how the search application derives meaning of the word 'Striped' and tries to improve the Ontology autonomously. Figure 5.1 is a snapshot of Ontology being used for this application. Figure 5.2 shows the state of properties of Striped Dolphin individual through Ontoviz editor before the user actually fires any query.

As soon as the user clicks Search button, a search for 'Striped' word is done in Ontology through the Protege-OWL API. The Protege expands user query and internally provides information to the Google API that 'Striped is a kind of Dolphin which is a Mammal'. The Google API returns an object with web search results. Figure 5.3 shows initial search result page for the query 'Striped'.

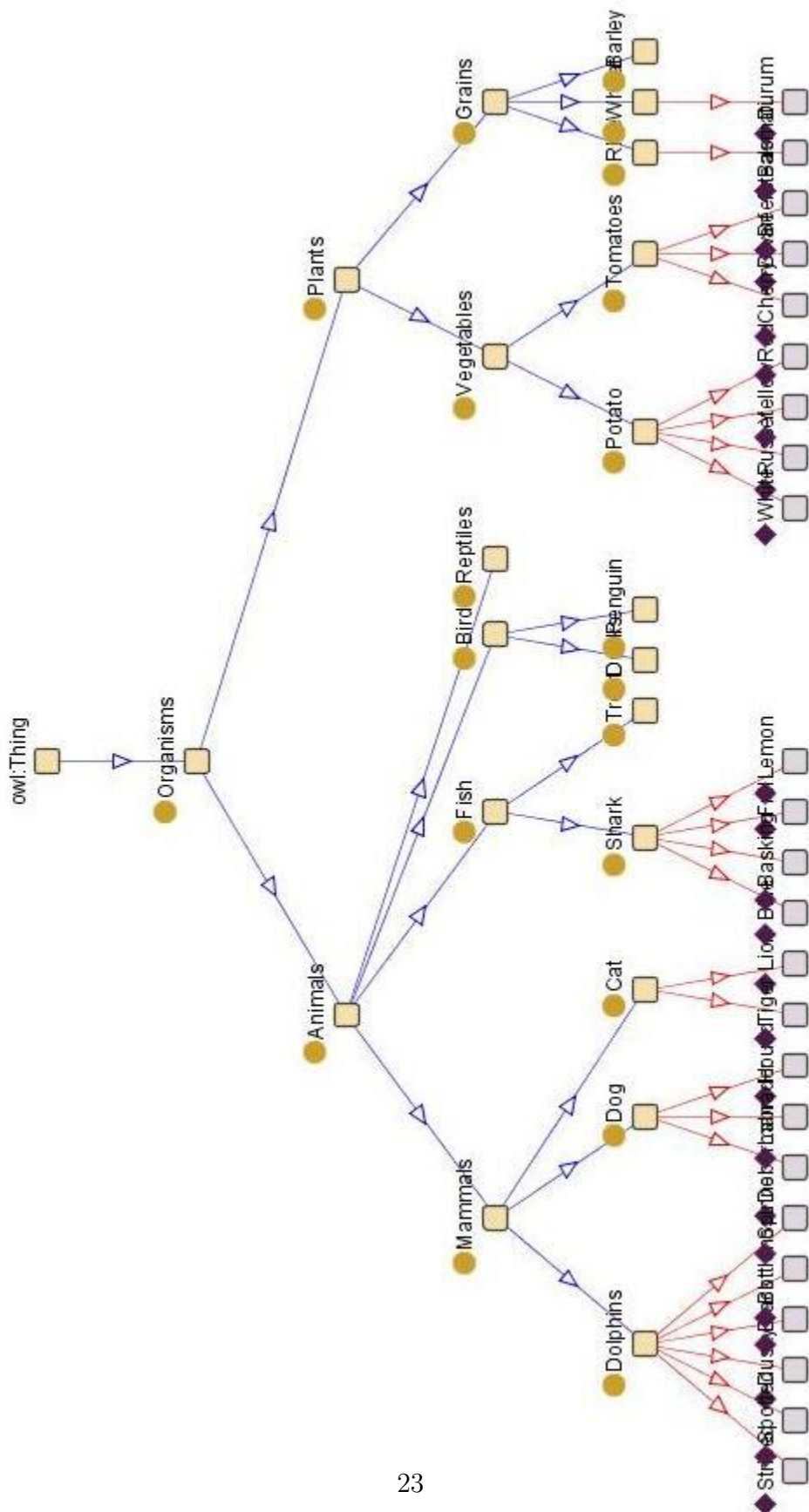


Figure 5.1: Ontology used for Semantic Search

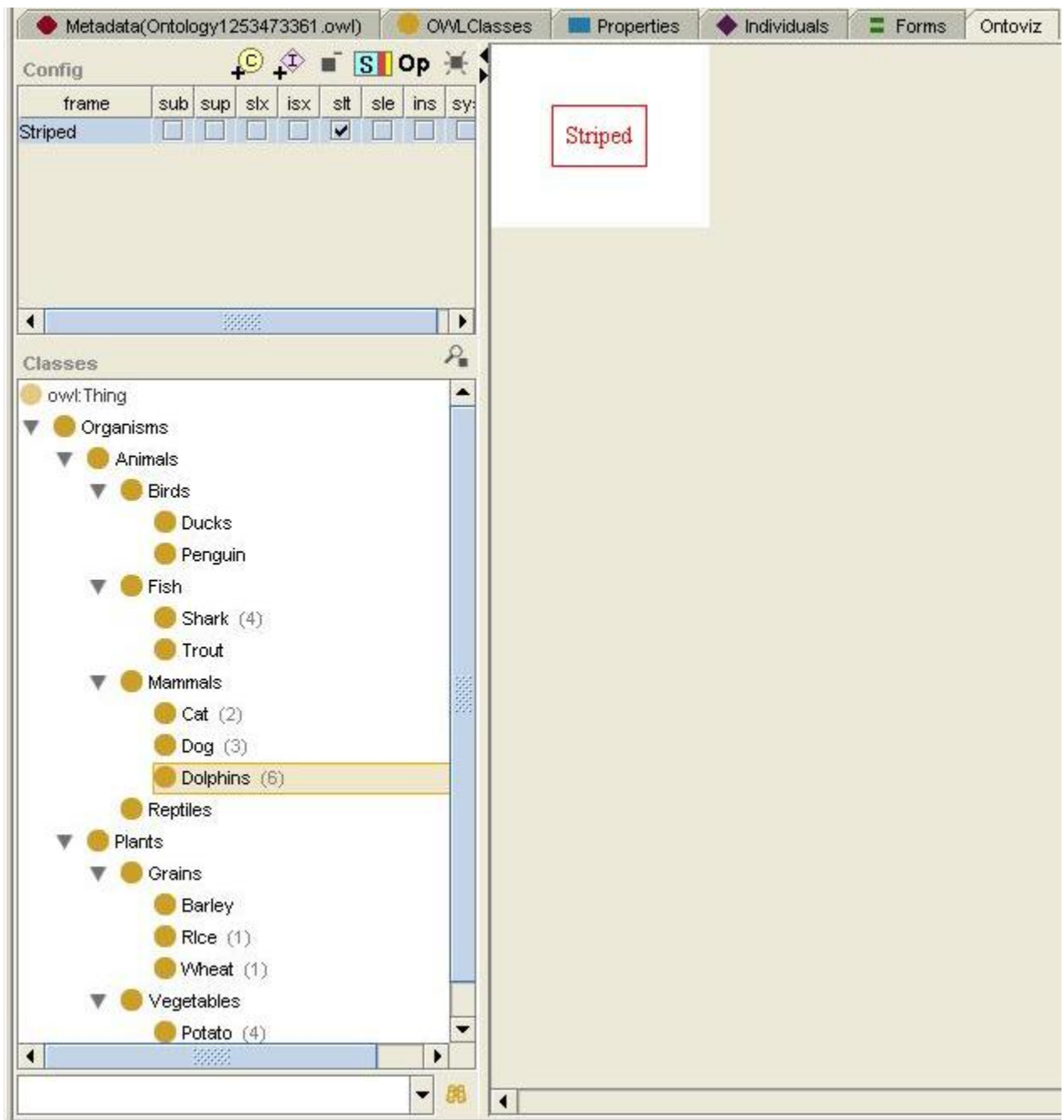


Figure 5.2: Striped Individual before search

The screenshot shows a web browser window with the address bar containing `http://localhost:8181/GoProSemanticWeb/Begin.do`. The page title is "Semantic Search". Below the title is an orange paw print icon. A search input field and a "Search" button are present. The search results are as follows:

- Striped Dolphin** - [Wikipedia, the free encyclopedia](#)  
The **Striped Dolphin** (*Stenella coeruleoalba*) is an extensively studied **dolphin**  
[en.wikipedia.org](#)
- Striped Dolphin** (*Stenella coeruleoalba*) - [Office of Protected ...](#)  
In the eastern tropical Pacific, **striped dolphins** are also called "streakers" by fi  
[www.nmfs.noaa.gov](#)
- [Whale Web: Striped Dolphin](#)  
Aug 5, 2000 ... **Striped dolphins** are often found in warm waters, usually stayin  
Atlantic ...  
[www.whale-web.com](#)
- Striped Dolphin**, *Stenella coeruleoalba* at [MarineBio.org](#)  
**Striped Dolphin**, *Stenella coeruleoalba*, Mammalia, Cetacea, Delphinidae, Des  
...  
[marinebio.org](#)

At the bottom of the results, there is a pagination link: **1 2 3 4 5 6 7 8** [More results »](#)

Figure 5.3: Search result for Striped Dolphin



Before we go on to discuss further steps, let's have a look at Figure 5.4 which shows the actual Google search results for the query 'Striped'.

As like normal Google search, a user can go on making new searches and browse the results with Semantic search. As the user keeps making new search, results of previous searches are simultaneously processed by the application. The first four search results are parsed by the HTML parser and ranked based on frequency of words matching the query. The web page which is ranked first is selected and processed by the Stanford Lexparser to get more information about Striped Dolphin. The new information is added to Organism Ontology using methods and inference engine of the Protege-OWL API. Figure 5.5 shows properties added to Striped individual.

Once the new information is added to the Ontology if the user types in same query sometime later into the search textbox, the application tries to give him direction into what would he like to search about that particular Animal or Plant (in this case it's 'Striped Dolphin') by reading out the updated Ontology and presenting it with checkbox on a JSP page. JSP page runs a java scriptlet which retrieves data using the Protege-OWL API and creates dynamic checkboxes on webpage. These properties are placed above the search results. Figure 5.6 shows properties added to the webpage. From the properties marked with a black rectangle we can see that the information retrieved is not irrelevant to something called 'Striped Dolphin'. For instance, first checkbox suggests that Striped Dolphins are capable of diving. Second checkbox suggests Striped Dolphin gives birth to calf. Third checkbox marked with rectangle has caught scientific name of Striped Dolphin. Other checkboxes indicate Striped Dolphin has melon (a typical upper body shape of a whale or Dolphin which can be seen above water surface) and is observed often or breaching(i.e. diving in this context).

The user can select any of the options available and the search application

Web Images Videos Maps News Shopping Gmail more ▼

Google

Striped

Search

Web [+ Show options...](#)

Results

[striped](#) - Definition from the Merriam-Webster Online Dictionary

Flash Player Required. The Adobe Flash Player is required in order to view this content. You can get it here: Get Adobe Flash player. Search "**striped**" in: ...  
[www.merriam-webster.com/dictionary/striped](http://www.merriam-webster.com/dictionary/striped) - [Cached](#) - [Similar](#)

[Striped Backgrounds](#)

**Striped Backgrounds** makes free **stripe** backgrounds, beautify your iPhone or Computer with a new free **striped** background.  
[stripedbgs.com/](http://stripedbgs.com/) - [Cached](#) - [Similar](#)

[Striped bass](#) - Wikipedia, the free encyclopedia

The **striped bass** (*Morone saxatilis*, also called stripers, rock , pimpfish or rockfish) is the state fish of Maryland, Rhode Island, South Carolina, ...  
[en.wikipedia.org/wiki/Striped\\_bass](http://en.wikipedia.org/wiki/Striped_bass) - [Cached](#) - [Similar](#)

[Data striping](#) - Wikipedia, the free encyclopedia

In computer data storage, data **striping** is the technique of segmenting logically sequential data, such as a single file, so that segments can be assigned to ...  
[en.wikipedia.org/wiki/Data\\_striping](http://en.wikipedia.org/wiki/Data_striping) - [Cached](#) - [Similar](#)

[+ Show more results from en.wikipedia.org](#)

[Striped Wall](#)

**Striped Wall**. A Screenshot Collective || The Gallery || · Supernatural 3×12 "Jus in Bello" ...  
Treat Yourself! **Striped Wall** is proudly powered by WordPress.  
[www.stripedwall.com/](http://www.stripedwall.com/) - [Cached](#) - [Similar](#)

[Stripe Generator](#) - [diy design](#) [stripes](#) background designer

Figure 5.4: Google search result for keyword 'Striped'

Striped	
http://www.owl-ontologies.com/Ontology1253473361.owl#have =	melon
http://www.owl-ontologies.com/Ontology1253473361.owl#Dolphin =	Programs
	coeruleoalba
	Laws
	Office
http://www.owl-ontologies.com/Ontology1253473361.owl#Photo =	dolphins
	Dolphin
	coeruleoalba
http://www.owl-ontologies.com/Ontology1253473361.owl#capable =	Did
	diving
http://www.owl-ontologies.com/Ontology1253473361.owl#dolphins =	ft
http://www.owl-ontologies.com/Ontology1253473361.owl#associate =	become
http://www.owl-ontologies.com/Ontology1253473361.owl#give =	rarely
	species
http://www.owl-ontologies.com/Ontology1253473361.owl#Status =	calf
	autumn
	summer
	birth
http://www.owl-ontologies.com/Ontology1253473361.owl#observed =	Dolphin
	coeruleoalba
http://www.owl-ontologies.com/Ontology1253473361.owl#Map =	often
	breaching
http://www.owl-ontologies.com/Ontology1253473361.owl#Range =	Dolphin
	Range

Figure 5.5: Properties added to Striped Individual



Figure 5.6: Properties shown on webpage

would make a new search which would then retrieve new set of results and hence new information regarding 'Striped Dolphin'. Lets assume that user selects a checkbox which says 'have[melon]'. Figure 5.7 shows the results after selection of 'have[melon]' property. These results contain information regarding Striped Dolphin and its melon. This becomes a new set of webpage results for the application to parse and retrieve information.

As the search continues, a new set of webpages help in addition of new properties to the individual Striped Dolphin in Organism ontology. Hence, next time when the user types in Striped as a query, he can see addition of new properties on the webpage which could help him to direct his search. Figure 5.8 shows addition of these new properties on webpage. Newly added properties are shown in black rectangle. First property indicates different regions in which Striped Dolphins could be found. Second property indicates what kind of environment or water they prefer. Upwelling here is the cold current of water coming from bottom of sea. Striped Dolphins prefer to stay in these areas of ocean.

On similar lines, we can see the search results for user query 'Cricket' in Figure 5.9. After successive search made by the user, we could see new properties added for Cricket Individual. Figure 5.10 shows new properties added to the webpage.

Another example would be to search for query 'Latex' as shown in Figure 5.11. We can see that all the Google search results are related to the LaTeX document editor whereas Semantic search results are related to the Latex plant. Newly added properties are shown in Figure 5.12. Properties marked in black rectangle determine some quality information related to the Latex plant.

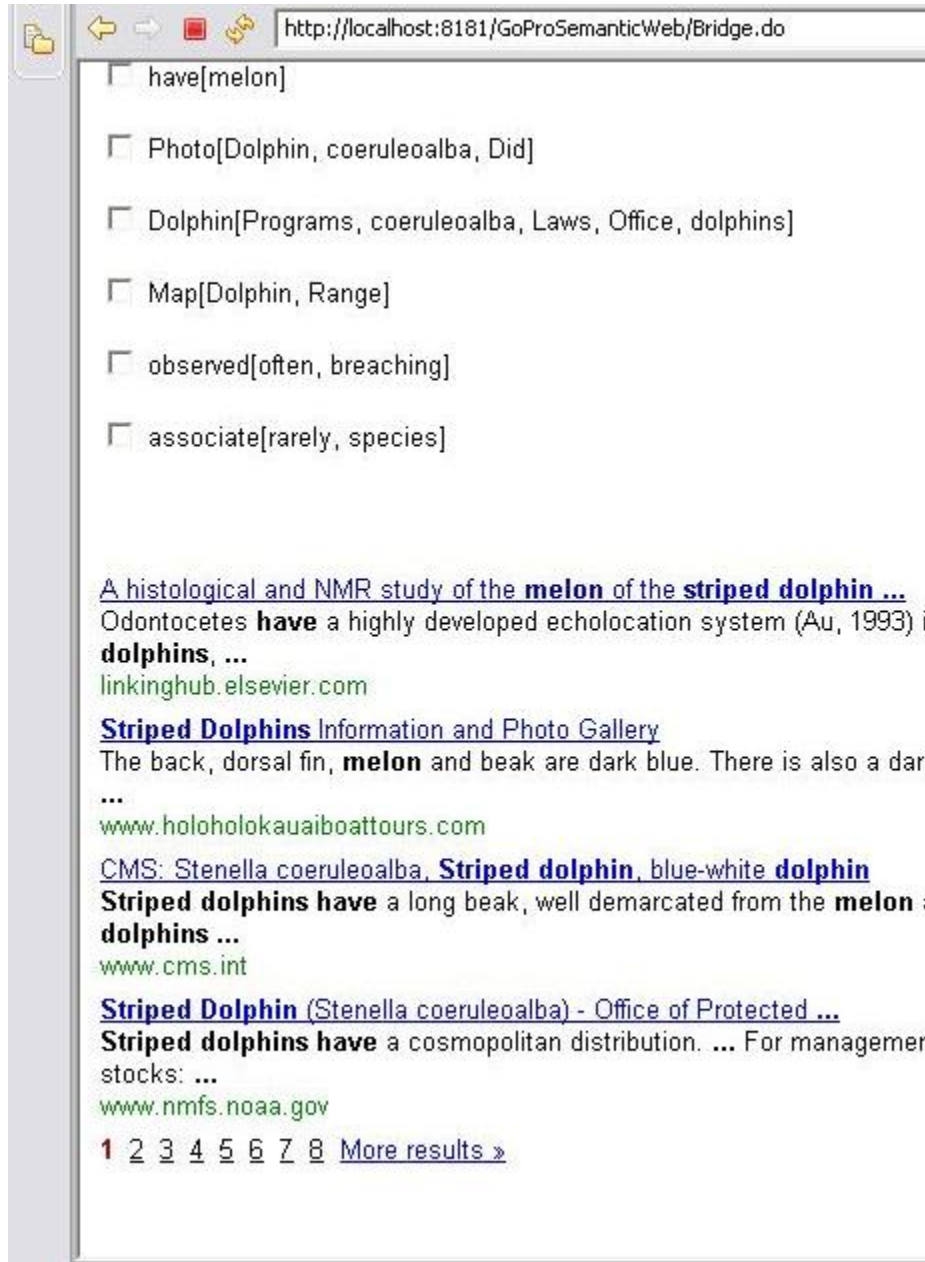


Figure 5.7: Results after selection of 'have[melon]' property



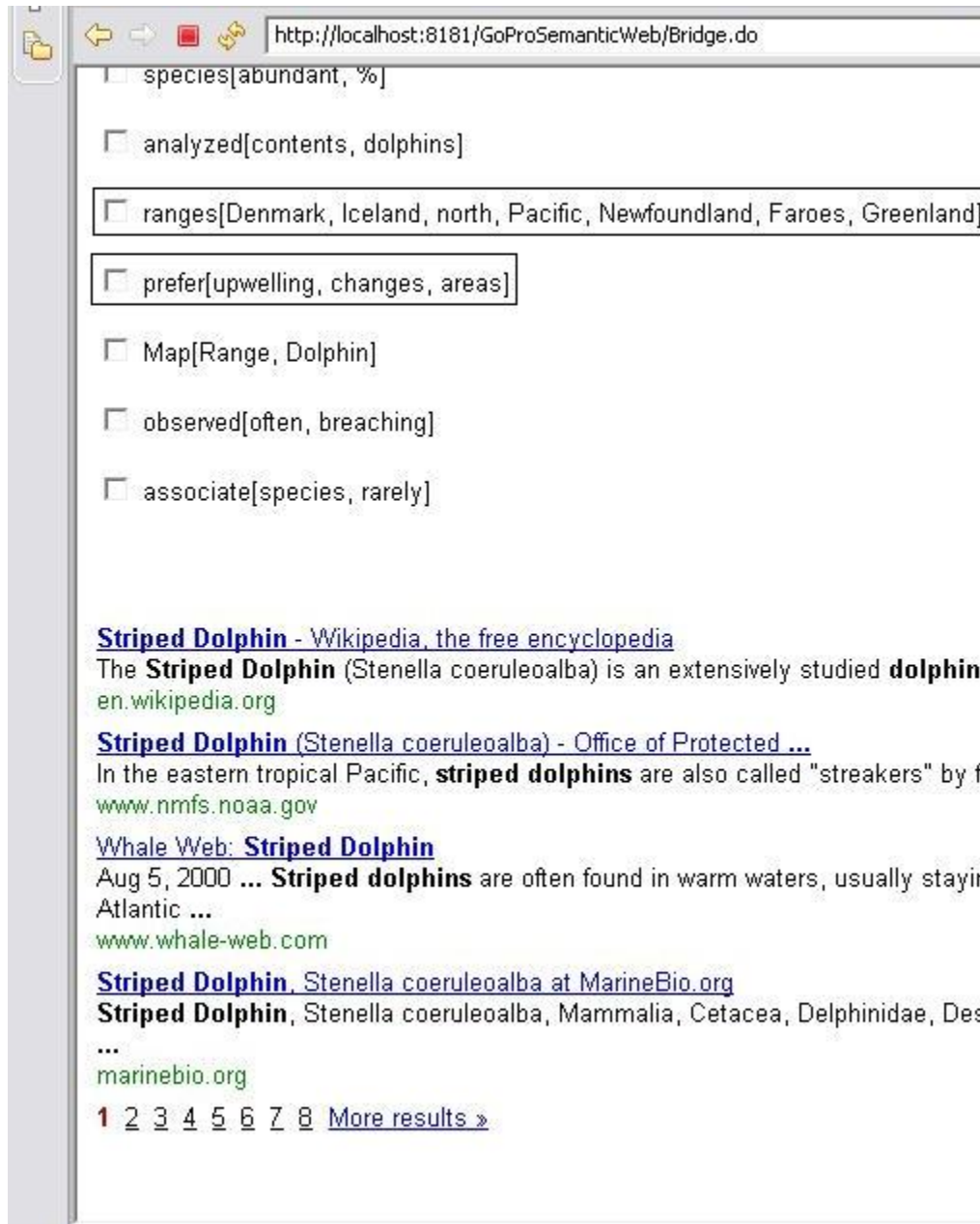




Figure 5.8: Addition of new properties



## Semantic Search




---

**Web** [+ Show options...](#)

**Cricket® Official Website**  
[MyCricket.com](#) \$25/Month Unlimited Talk & Text Simpl

New Zealand vs. Pakistan, 1st T20, 12th Nov '09



Pak 161/8(20) | NZ 42/4(6.3)  
 NT Broom\* 1(1), BJ Watling 3(8)  
 NZ needs 120 to win off 13.3 overs Required RR 8.6  
 Last 2 overs: ... 1 4 ... | 1 W 1  
[ws-4.willow.tv](#)  
[Cricbuzz](#) [Cricinfo](#) [Willow](#)

**Cricket Wireless | Cell Phones, Cellular Phone Pl**  
**Cricket** Wireless is a leading cell phone provider, offering a unlimited minutes and no signed contract.  
[www.mycricket.com/](#) - [Cached](#) - [Similar](#)

<a href="#">Phones</a>	<a href="#">Plans</a>
<a href="#">Send Text</a>	<a href="#">PAYGo</a>
<a href="#">Broadband</a>	<a href="#">Coverage</a>
<a href="#">Find A Store</a>	<a href="#">Downloads</a>

[More results from mycricket.com »](#)

Prepaid Cell Phones | **Cricket** Wireless | Pay As U  
 The **Cricket** CAPTR is the low cost camera phone you've b capture the moment with **Cricket** CAPTR! Camera w/ zoom  
[www.mycricket.com/cricketphones/](#) - [Cached](#) - [Similar](#)

[Cricinfo.com](#) | **Cricket** news, live scores and stati:

[Cricket \(insect\) - Wikipedia, the free er](#)  
**"Cricket** singing means rain: semiotic n  
[en.wikipedia.org](#)

[cricket \(insect\) -- Britannica Online Enc](#)  
 Britannica online encyclopedia article or  
[www.britannica.com](#)

[cricket \(insect\) Facts, information, pict](#)  
 Jul 12, 2009 ... Get information, facts, a  
[www.encyclopedia.com](#)

[Facts about Cricket Insect](#)  
 Nov 2, 2007 ... **Cricket insects** may be  
[www.buzzle.com](#)

**1** [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [More results »](#)

Figure 5.9: Search results for query 'Cricket'



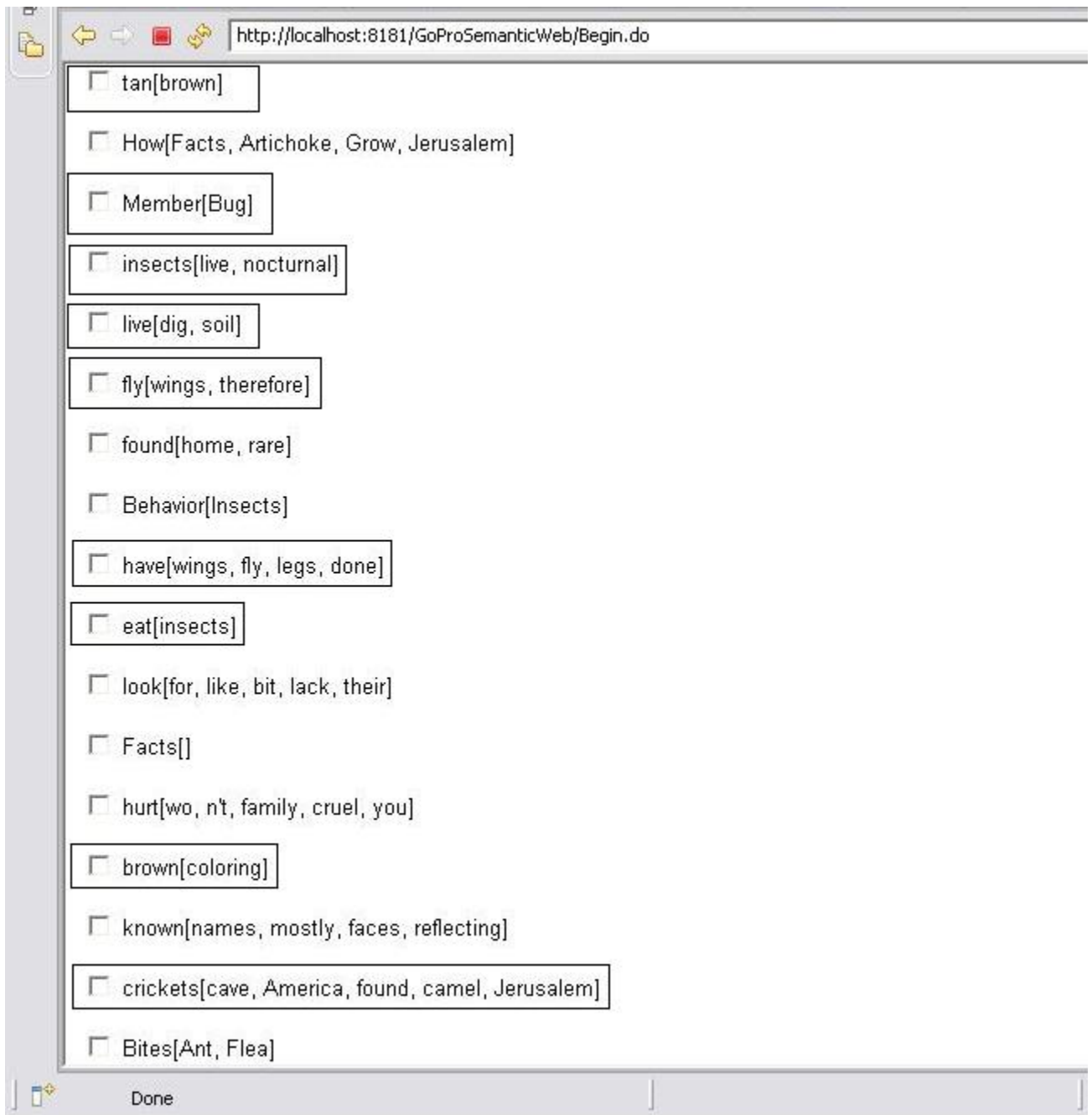


Figure 5.10: New properties added to Cricket Individual

The image shows a side-by-side comparison of search results for the query 'Latex'. On the left is a standard Google search interface, and on the right is a 'Semantic Search' interface.

**Google Search Results:**

- Search bar: Latex
- Web  Show options...
- LaTeX project: LaTeX** – A document preparation system. The latest version of LaTeX and information on the experimental code. [How to get it](#) - [Documentation](#) - [Introduction](#) - [The LaTeX project](#) - [www.latex-project.org/](#) - [Cached](#) - [Similar](#)
- LaTeX project: An introduction to LaTeX** Jun 10, 2009 ... LaTeX is a document preparation system most often used for medium-to-large technical documents. [www.latex-project.org/intro.html](#) - [Cached](#) - [Similar](#)
- Show more results from [www.latex-project.org](#)
- LaTeX** - Wikipedia, the free encyclopedia LaTeX is most widely used by mathematicians, scientists, economists and other scholars in academia and the publishing industry. [Typesetting system](#) - [Example](#) - [Pronouncing and using LaTeX](#) - [en.wikipedia.org/wiki/LaTeX](#) - [Cached](#) - [Similar](#)
- TeX Users Group (TUG) home page** It is nearly entirely member-supported, so if you use ConTeXt, Metafont, MetaPost, Texinfo, et al. ... [www.tug.org/](#) - [Cached](#) - [Similar](#)
- MiKTeX Project Page** This is the MiKTeX project page. MiKTeX is a type system. The distribution includes TeX, pdfTeX and miktex.org/ - [Cached](#) - [Similar](#)

Navigation: Find: exposure   Done

**Semantic Search Results:**

- Search bar: Latex
- Latex** - Wikipedia, the free encyclopedia Similarly, it may provide some protection against brown rot. [en.wikipedia.org](#)
- Houseplants and Latex Allergy: Can Your Plants Be Safe?** Dec 8, 2008 ... People who suffer from latex allergies should avoid indoor plants. [www.associatedcontent.com](#)
- Plant Defense-Related Proteins as Latex Allergens** Cross-reactive allergens in higher plants. In the Division of Allergy and Immunology, National Institute of Health. [dmd.nihs.go.jp](#)
- Inquiries concerning the latex within plants** - Botany. I searched the web and see sth on this topic, it says that latex is a natural rubber. [forums.gardenweb.com](#)

Navigation:         [More results »](#) Done

Figure 5.11: Search results for query 'Latex'

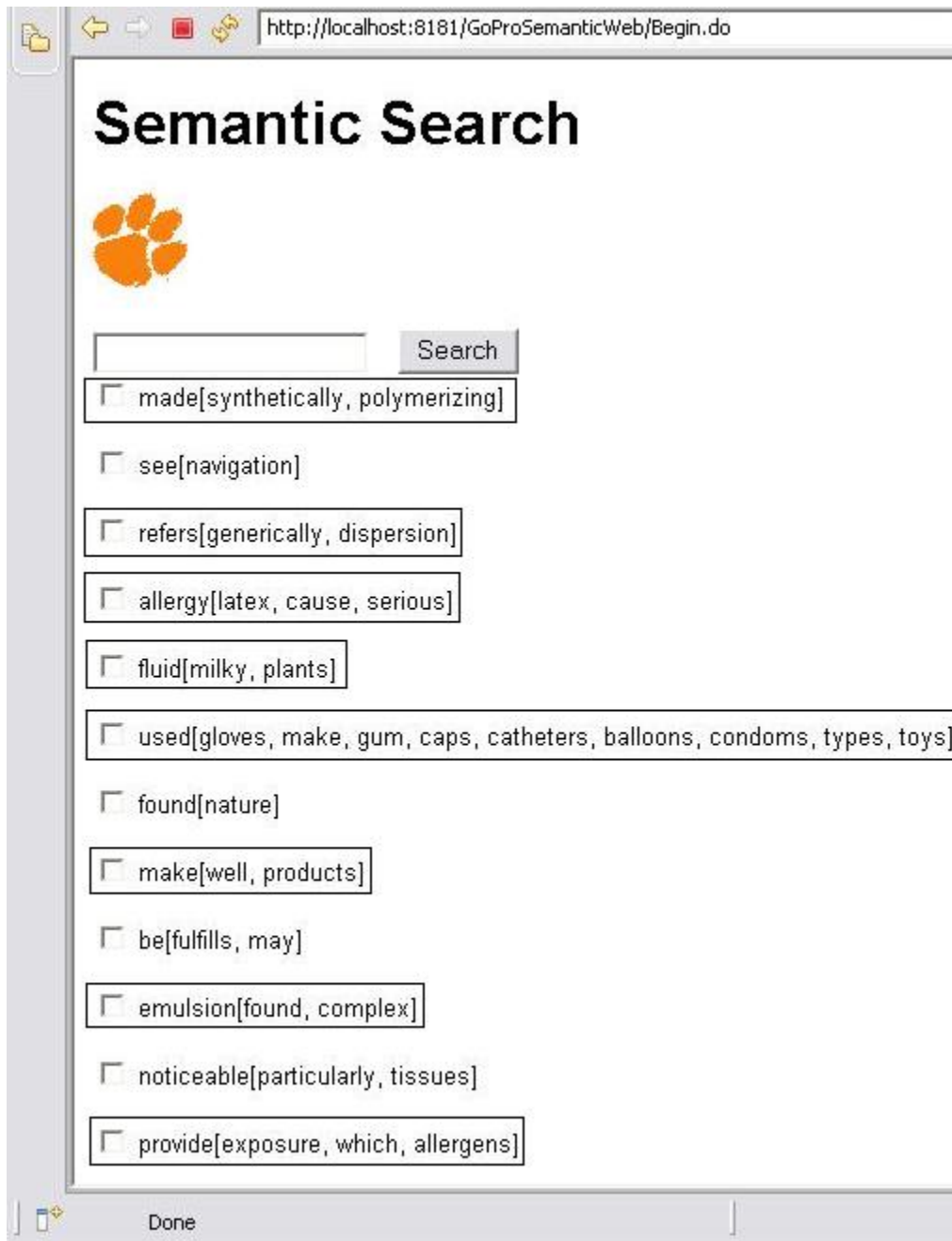


Figure 5.12: New properties added to Latex Individual

## 5.2 Code description

1. Model Package: `com.research.model`
  - (a) `Clips.java`: Imports the CLIPSJNI and provides access to COOL functions.
  - (b) `Lexparser.java`: Derives grammatical relations between words and updates the OWL model using the Protege-OWL API and the Stanford Lex-Parser.
  - (c) `ParseUrl.java`: Extracts text from webpages using the HTML parser.
  - (d) `OWLhandler.java`: Provides methods to update, query, validate and create OWL model.
2. Controller Package: `com.research.controller`
  - (a) `Begin.java`: Provides communication between the View and the Model when the user does search for the first time.
  - (b) `Bridge.java`: Provides communication between the View and the Model during query expansion and OWL model update.
3. View: Java src Webcontent
  - (a) `index.jsp`: Provides initial user interface for the first query and communicates with `Begin.java` Servlet.
  - (b) `response.jsp`: Provides user interface for search, runs the Google Web Search API, creates dynamic checkbox options and communicates with `Bridge.java` Servlet.

# Chapter 6

## Conclusions and Discussion

### 6.1 Semantic Search

Slowly and steadily the Semantic Web is becoming the way people seek online solutions. Semantic Web adds meaning to our online activities and hence it is more useful than just any other non-semantic application. Apart from applications like Semantic Social networks and Semantic online meetups, Semantic search is just another application of Semantic Web. The key point of this research is not just running through webpages with Semantics attached but, updating the current Ontology autonomously to guide the user or help him direct his search. This research was evolved in three parts.

1. Semantic Search with the CLIPS.
2. Semantic Search with the Protege API.
3. Semantic Search with the Protege API and an Ontology improvement through NLP and Parser.

The Protege API approach clearly scores over the CLIPS due to time and .clp to .owl file change factor. The technology and tools used in this research are:

1. Technology: Java Servlets and JSP.
2. IDE: Eclipse.
3. Server: Apache TOMCAT 6.0.
4. Inference engine: CLIPS, Protege API.
5. Parser: NLP Stanford Lexparser, HTML parser.
6. Other tools: CLIPSOWL.
7. Search engine API: Google API.
8. Configurations: OS Windows XP, 2.5 GB RAM, Intel Core Solo Processor T1350 1.86 GHz.

## 6.2 Further scope of study

As we have seen, there are two major shortcomings of this application which could pave the way for further improvement and study of this application.

1. One major drawback of the Stanford Lex-Parser is the time it takes to parse large chunk of text thrown at it. The Semantic search application ranks web-pages and chooses only the first ranked webpage for the Stanford Lex-Parser processing. Whole process of extracting and updating properties with new web search results takes on an average 2-3 minutes to complete. This time delay can be improved to a certain extent by parallel processing. Task of parsing of

webpages and extraction can be further divided among processors to expedite the whole process.

2. Some of the properties attached to an Individual seem inaccurate. A larger dictionary to exclude probable meaningless relations and words would help in this matter. But a larger dictionary will end up slowing the whole process. Hence, there is a trade off between accuracy of results and speed of applications. Another approach would be to provide the user to choose and deselect the properties which he/she thinks are meaningless or irrelevant.
3. If we have a look at the results of autonomously generated properties of 'Cricket' and 'Latex' individuals we see that the accuracy rate of meaningful results lie between 52-66 percent. A good result or set of properties would be something around 60 percent as of now. This accuracy can be increased by adding a filter to our results.
4. Following the previous point, we can have either autonomous or manual filtering of those properties which seem meaningless. For instance, some of the autonomously derived properties like look[for, like, bit, lack, their] make no sense for the user to select and proceed for search related to 'Cricket' as a subject of search. This process of cleaning up bad or meaningless properties can be incorporated as a further part of enhancement to the application.
5. There is a trade-off between query enhancement and accuracy of results. After the user selects a property checkbox and proceeds with search related to a subject, our application combines the subject, its parent class information and its property values to enhance the query and get the search results. If the property has multiple values then there is a chance of getting a defocused search. It is

also true in the case where property has one value but parent class information is more. Maximum accuracy of results is seen with a triplet of a superclass name, a subject and a property value. Hence, as a part of further scope, the selection of single best property value can be one of the solutions to save the search from getting defocused.



# Bibliography

- [1] Rokia Bendaoud, Mohamed Rouane Hacene, Yannick Toussaint, Bertrand Delecroix, and Amedeo Napoli, "Text-based ontology construction using relational concept analysis" *International Workshop on Ontology Dynamics, France*, June 2008.
- [2] Alexander Maedche and Steffen Staab, "Discovering Conceptual Relations from Text" *International Workshop on Ontology Dynamics, Germany*, 2000.
- [3] Krishnakumar Pooloth and Raoul Jetley, "Integrating the CLIPS Rule Engine with Protege" *AIML 05 Conference, CICC, Cairo, Egypt*, December 2005.
- [4] Wenjie Li, Zhiyong Feng, Yong Li and Zhoujun Xu, "Ontology-based Intelligent Information Retrieval System", *IEEE Canadian Conference, Canada* 2004.
- [5] Sunil Movva, Rahul Ramachandran, Xiang Li, Phani Cherukuri and Sara Graves, "Noesis: A Semantic Search Engine and Resource Aggregator for Atmospheric Science" *NSTC Conference, USA*, 2007.
- [6] Harith Alani, "Position Paper: Ontology Construction from Online Ontologies" *International World Wide Web Conference, Edinburgh, Scotland*, 2006.
- [7] Hai H. Wang, Natasha Noy, Alan Rector, Mark Musen, Timothy Redmond, Daniel Rubin, Samson Tu, Tania Tudorache, Nick Drummond, Matthew Horridge and Julian Seidenberg, "Frames and OWL Side by Side" *9th International Protege Conference, Stanford, California*, 2006.
- [8] Paul R. Smart, Nigel R. Shadbolt, Leslie A. Carr and Monica C. Schraefel, "Knowledge-Based Information Fusion for Improved Situational Awareness", *8th International Conference on Information Fusion, Philadelphia, USA*, July 2005.
- [9] Danushka Bollegala, Yutaka Matsuo and Mitsuru Ishizuka, "Measuring Semantic Similarity between Words Using Web Search Engines", *16th International World Wide Web Conference, Alberta, Canada*, May 2007.
- [10] The Protege-OWL API, <http://protege.stanford.edu/plugins/owl/api/guide.html>.

- [11] The Protege-OWL Editor, <http://protege.stanford.edu/overview/protege-owl.html>.
- [12] The Protege-Frames Editor, <http://protege.stanford.edu/overview/protege-frames.html>.
- [13] The Stanford Lex-Parser, <http://nlp.stanford.edu/>.
- [14] The HTML parser, <http://htmlparser.sourceforge.net/>.
- [15] The CLIPS Expert System, <http://clipsrules.sourceforge.net/>.
- [16] The CLIPSOWL, <http://groups.google.com/group>.
- [17] The Pellet DL reasoner, <http://clarkparsia.com/pellet>.
- [18] The OWLAPI, <http://owlapi.sourceforge.net/>.
- [19] The Google AJAX Web Search API, <http://code.google.com/apis/ajaxsearch/>.