SEW
The Semantic Extensions to Wikipedia

A Thesis Presented to
The Faculty of the Computer Science Program
California State University Channel Islands

In (Partial) Fulfillment
of the Requirements for the Degree
Masters of Science in Computer Science

By
Subha Krishnan
March 2007

APPROVED FOR THE COMPUTER SCIENCE PROGRAM

_____

Advisor: Dr. Andrzej Bieszczad Date

_____

Advisor: Dr. William Wolfe Date

_____

Advisor: Dr. Peter Smith Date

APPROVED FOR THE UNIVERSITY

_____

Advisor: Dr. Gary A. Berg Date

SEW

The Semantic Extensions to Wikipedia

by

Subha Krishnan

Computer Science Program
California State University Channel Islands

Abstract

The Semantic Web is the next step in the evolution of the Web. The goal of the Semantic Web initiative is to create a universal medium for data exchange where data can be shared and processed by people as well as by automated tools. The thesis presents **SEW** (**S**emantic **E**xtensions to **W**ikipedia), a system that uses the Semantic Web technologies to extract information from the user and store the data along with the semantics. An experiment is carried out and SEW is evaluated for its performance in the real world domain. Semantic Extensions to Wikipedia will serve as a model website empowered by the Semantic Web that would encourage other developers to integrate this powerful technology into their own web applications.

Acknowledgements

I am grateful to my advisor, Dr. Andrzej Bieszczad for his support, advice and guidance. His suggestions were very useful in resolving the difficulties that I faced in completing my thesis. I also thank my husband, Sampath Krishnan for supporting and encouraging me to do my best. Finally, I am grateful to my father, Mr. Raghunathan for his interest and guidance that helped me complete my thesis.

Table of Contents

1. Introduction……..7
   - 1.1 Today's World Wide Web....7
   - 1.2 The Semantic Web....9
   - 1.3 Moving from the current Internet into the Semantic Web....10
   - 1.4 Semantic Extensions to Wikipedia....11
   - 1.5 Overview of the Thesis Document....11

2. Study of existing Semantic Web technologies and projects....13
   - 2.1 Semantic Web technologies....13
     - 2.1.1 Data representation with XML....13
     - 2.1.2 RDF and RDFS....14
     - 2.1.3 OWL....15
   - 2.2 Semantic Web projects....17
     - 2.2.1 Infrastructure....18
     - 2.2.2 Knowledge Extraction....18
     - 2.2.3 Construction of Ontology and Knowledge Base....19
     - 2.2.4 Applications of an Ontology....21
     - 2.2.5 Other projects....22
   - 2.3 Study of tools for building ontology based applications....23
     - 2.3.1 Jena....23
     - 2.3.2 Pellet....23
     - 2.3.3 IBM Integrated Ontology Development Toolkit....24
     - 2.3.4 WebOnto....24
     - 2.3.5 Protégé....24
     - 2.3.6 KAON....24

3. Analysis of Semantic Extensions to Wikipedia....25
   - 3.1 Use Case Diagram....26
   - 3.2 System Architecture....27
   - 3.3 Process Flow....28
     - 3.3.1 User uploads an Ontology....28
     - 3.3.2 User browses through existing ontologies....29
     - 3.3.3 User wishes to enter a new record in the knowledge base for a given ontology……....29
     - 3.3.4 User wishes to edit the knowledge base....30

4. Implementation of the SEW....31
   - 4.1 Examination of software tools used in the project....31
     - 4.1.1 Eclipse....31
     - 4.1.2 MySQL....31
     - 4.1.3 Tomcat....32
     - 4.1.4 Java....33

4.1.5 RdfReactor 34
4.1.6 Velocity 34
4.2 Code walkthrough 35

5. Experimenting with the SEW 40
5.1 Building the ontologies 40
5.1.1 The Course ontology 40
5.1.2 The Program ontology 41
5.1.3 Association Between Program And Course ontology 42
5.2 Uploading the ontologies 42
5.3 Creating the knowledge bases 45
5.3.1 The Course knowledge base 45
5.3.2 The Program knowledge base 47
5.3.3 The AssociationBetweenProgramAndCourse knowledge base 47
5.4 Analysis of results 50

6. Future Work 52
6.1 Future Research and Implementation Work 52

7. Conclusions 53

Chapter 1

Introduction

1.1 Today's World Wide Web

The World Wide Web is a collection of documents stored on Internet servers. Users can access the web documents over the Internet using a web browser to obtain information on a variety of topics.

HTML (Hypertext Markup Language) is used to create and design web pages. It became very popular through the 1990s. With the advent of WYSIWYG (What You See Is What You Get) editor, it became easy even for a novice user to render content over the Internet using HTML's loose syntactic rules.

The World Wide Web has rapidly grown over the last decade and as of date, 5 February 2007, it contains approximately 13.8 billion pages [1]. With the rapid growth, HTML's limited expressive power to represent information started falling short.

The limitations of information creation and extraction within the Internet are as follows:

1. Software Agents

A software agent in an Internet domain can effectively search, find and filter information from the Web and bring it to a user's attention. However to access the information, the agent has to parse the HTML making it dependent on the web page implementer [2]. When the design of the web page changes, the original parsing does not work and needs to be modified.

Consider the following two common ways of displaying product details using the tabular format available in HTML,

Table 1

Name	*Price*	*Quantity Available*
Product A	100	300
Product B	200	200
Product C	300	100

Table 2

Name: Product A	*Price*: 100	*Quantity Available*: 300
Name: Product B	*Price*: 200	*Quantity Available*: 200

If a user were asked what is the price of product B, he would answer 200 using either of the two tables. However an agent browsing the Internet in search of the price faces a problem when it comes across the above tables. HTML structures data but offers no assistance regarding the semantics of data. So the agent does not know whether to look at data that is adjacent or below 'Price'. Also, if the web page uses 'Cost' instead of 'Price', then the agent may ignore the required data.

Web pages, written in HTML are designed to be read by people. HTML, a presentation language provides us with fancy techniques (markup elements) for formatting and displaying data. To a software agent, that neither understands HTML nor English, processing this web page becomes very difficult.

2. Search Engines

Users can explore the vast amount of information available in the Internet using popular search engines like Google, Yahoo! Search and Windows Live Search.

Many search engines retrieve pages from the Web, index the words in each document, and store this information efficiently. When a user searches the Web using a search phrase the search engine determines all the pages on the Web that contain the words in the search phrase. Additional factors such as distance between the words in the search phrase can also be taken into consideration during the search.

However since 95% of the text in web pages is composed from around 10,000 words (a rough estimate) most searches will return a huge number of pages for the given search criteria. Google uses the Page Rank algorithm to rank the importance of the pages that fit the search criteria and thereby displays the more important pages to the user at the top. A page that is linked to by many pages with high PageRank receives a high rank itself. If there is lesser number of links to a web page it receives a lower PageRank [3].

The underlying keyword search is responsible for frequently returning irrelevant answers or 'false hits'. If we search Google with the keyword ' apple', it returns links that contain the latest news, product information etc., about 'Apple Computer Inc'. This may be frustrating to the user who just wants to find some information about the fruit, apple. Users need to keep modifying their search criteria till they get relevant results.

Search engines can perform better by including a context-based search. If the web pages can include the context in which terms appear in the document, the search would boil down to a modified keyword search. Context could be used to determine the relevancy of a particular document in a specified search.

3. Wikis

A Wiki is a type of web site that allows the users to add, remove and edit available information, sometimes without the need for registration. The ease of interaction makes a Wiki an effective tool for collaborative authoring [4].

Wikipedia is a successful implementation of a Wiki. Users can author articles on a variety of topics. Other users can access and edit the information. Its simple and straightforward approach makes it very popular. However its search capability is limited. Also if the user has a question, he has to search for relevant documents, read through the

retrieved documents and extract the required data. Thus human intervention is necessary to use Wikipedia.

If the information in Wikipedia is accompanied by metadata, then a software program can reason about the data and return answers in response to a user's question. Documents could be used to build domain-specific knowledge, which in turn can be analyzed and reused. However metadata representation requires appropriate structures, a feature not yet available in HTML.

1.2 The Semantic Web

The Semantic Web [6] provides a common framework that allows data to be shared and reused across applications by putting together documents with computer-processable meanings (meta-level data) on the World Wide Web. The Semantic Web extends the Web by using knowledge representation technologies.

The overview of the various components in the Semantic Web architecture [5] is given below. The technical details will be discussed in the next chapter.

1. Knowledge Representation

The technologies for representing knowledge in the Semantic Web are:

a. XML

XML allows users to apply a tree-based structure to information in a machine-readable format. XML Schema can be used to express a set of rules that the XML document must conform to for it be considered as valid.

b. RDF

RDF is a simple data model for referring to objects (resources) and how they are related. A RDF-based model is represented in XML syntax. Data can be supplemented by metadata. If 805-556-7890 is the data, then the metadata can add context to the data by indicating what the data means, in this case by stating that the data is a telephone number. Machines can then infer relationships between resources through the metadata. RDF Schema (RDFS) is a vocabulary for describing properties and classes of RDF resources, with semantics for creating hierarchies of properties and classes.

2. Ontologies

There is a need for the machine to understand the meaning of the various metadata definitions. Ontology is a vocabulary that contains a list of terms that have been enumerated explicitly and have an unambiguous, non-redundant definition. It includes a grammar for using vocabulary terms and has formal constraints on how terms in the ontology's controlled vocabulary can be used together.

Ontologies can be developed using RDF Schema and Web Ontology Language (OWL). OWL adds more vocabulary for describing properties and classes. Some of them are relations between classes (e.g. disjointness), cardinality (e.g. exactly one), equality, richer typing of properties, characteristics of properties (e.g. symmetry) and enumerated classes [7].

3. Agents

The real power of Semantic Web will be realized when software agents collect data from different sources, exchange it with other agents and integrate and analyze this data to draw conclusions.

The agents will use a reasoning engine. SWRL is a proposal for a Semantic Web rules-language, combining sublanguages of OWL (OWL DL and Lite) with those of the Rule Markup Language to analyze data and answer questions.

1.3 Moving from the current Internet into the Semantic Web

The Semantic Web is the next step in the evolution of the Web. Currently, the World Wide Web is based on documents written in HTML. HTML is a data presentation language but not a data representation language. It consists of tags, to mark up a body of text interspersed with multimedia objects such as images. HTML determines how the data in the web documents is displayed to the user. The current Internet is highly scalable but there is a distinct need for content management and that is where the Semantic Web comes in.

The goal of the Semantic Web initiative is to create a universal medium for data exchange where data can be shared and processed by people as well as by automated tools. One way of achieving this goal is to employ artificial intelligence and train machines to behave like people. The main challenge in this case is how to equip the machine with a complete and powerful natural language processing utility. The same fact can be expressed in many different ways: ‘Roses are red and Violets are blue’. ‘The color of Rose is Red and that of Violet is Blue’, ‘Red Roses and Blue Violets’. Although the tasks of understanding these statements may seem trivial to a human being, to train the machine to interpret the meaning of simple English statements is a tall order to accomplish.

The Semantic Web approach tackles the problem by developing languages for expressing information in a machine processable form. Simple English statements are marked up by powerful XML tags that add meaning to the content. The tags could be machine-understandable information about the human-understandable content (such as the creator, title, description, etc., of the document) or it could be metadata representing a set of facts such as resources. Common metadata vocabularies and maps between vocabularies allow document creators to know how to mark up their documents so that there is a single standard for websites to publish relevant data. This will also ensure that software agents browsing the Web will not get confused between the ‘Rose’ flower and the ‘Rose’ Bowl Game.

As the Semantic Web has to describe different kinds of data, it is clear that the language employed has to be complete and very expressive. As stated earlier, the Semantic Web includes descriptive technologies like the data-centric, customizable Extensible Markup Language (XML), Resource Description Framework (RDF) and Web Ontology Language (OWL). These languages have rich vocabularies that allow the user

to specify the semantics of data and employ a strict schema to impose restriction on the structure and meaning of documents. XSLT (Extensible Stylesheet Language Transformations) is a template processor and can be used to convert a XML document into a HTML document for web browser rendering. Although the Semantic Web does not include an inference engine, various knowledge reasoning models and query engines can be built on top of the Semantic Web that can assist software agent navigating through the Web and reasoning about the available data. This process facilitates automated information gathering and research. Data coming from different Internet applications can be combined and analyzed.

1.4 Semantic Extensions to Wikipedia

Wikipedia is a web based portal. Anyone with access to the web site can create, edit and access the articles. The data within Wikipedia is displayed as factual documents in response to a user query. To perform analysis on data, Wikipedia, which consists of hyperlinked documents, needs to be structured differently. We need a data model composed of definitions and relations, and a knowledge base consisting of objects that conform to the model. The data model will encompass the different entities occurring in the world. Entities are defined by their properties and are related to each other. The model should be representative of the real–world and should promote reusability of knowledge.

In Semantic Extensions to Wikipedia project, we attempted to build the data model and knowledge base using the Semantic Web technologies. Semantic Web provides us with descriptive languages to describe data in a machine-processable format. The challenges in this project are

1. how to gather information from the user?,
2. how to build the data model? and
3. how to construct the knowledge base?.

Knowledge extraction should be done in a user friendly way as the success of the Semantic Extensions to Wikipedia (like Wikipedia) depends on user participation and contributions. All the technical details of the project should be hidden from the user and the user should be able to interact with the system transparently. Once the data model and knowledge base is in place, additional analysis techniques can be built on top of it.

1.5 Overview of the Thesis Document

In Chapter 2, we will discuss various existing Semantic Web technologies and projects. The success of the Semantic Web requires a necessary infrastructure, knowledge extraction tools, suitable data representation languages and processing tools. We will provide an overview of the work that is being done in these areas, so that it will give us an insight to what goals still need to be achieved and the work that needs to be done. Projects are classified into various categories and a short outline on their purpose, functionality and implementation is included.

In Chapter 3, we will perform analysis on Semantic Extensions to Wikipedia. In this chapter we will define the project, its goals and functionality. We will discuss the rationale behind the various design decisions. We will investigate the system architecture and process flow as well as discuss some of the issues that we faced at this stage and see how we worked around them.

In Chapter 4, we will discuss Semantic Extensions to Wikipedia project implementation. We will examine the software and tools used in the project. We will do a code overview to see how the various functionalities were achieved. The project is implemented in Java and follows the client-server model. We will discuss the various implementation issues and their resolutions and study RDFReactor, a project that allows a developer to implement Semantic Web applications in Java. Finally, we will examine the interfacing between Semantic Extensions to Wikipedia and RDFReactor. The highlight of this project is its transparency. The user is blissfully unaware of the background processes. Data is processed and transformed dynamically and this feature has not been hardcoded.

In Chapter 5, we will test Semantic Extensions to Wikipedia to evaluate its performance in the real world domain. We will build a test application, apply SEW within the application and draw observations.

In Chapter 6, we will discuss the future research and implementation work that can be done within the scope of the project. Also, we will discuss enhancements to the project that can be done to improve SEW's functionality.

In Chapter 7, conclusions from the thesis research, project implementation and the experiment performed will be presented. Adoption of the Semantic Web technology can radically change the way people use the Internet and redefine its purpose. This thesis is a step in that direction.

Chapter 2

Study of existing Semantic Web technologies and projects

In this chapter, we will first discuss the technologies involved in building the Semantic Web. We will see how the technologies evolved from a need to be more descriptive and better represent the real world. We will include the language specifications and introduce the vocabulary through examples. In the next part of the chapter, we will discuss the various Semantic Web research and implementation projects. Projects are classified based on the goals they are trying to achieve. The Semantic Web is a vast and active area of research and we will be able to understand the current status of the technology and its future challenges through this overview.

We will conclude the chapter with a short description of tools that provide features and functionalities useful in the implementation of our project.

2.1 Semantic Web technologies

This section includes a study of XML, RDF, RDFS and OWL. Each language is an extension of its predecessor, both, in the information that can be expressed and concluded.

2.1.1 Data representation with XML

XML [9] stands for Extended Markup Language. It is a language that is tailor-made for data representation. A XML and HTML document look very similar. In fact, it is possible to confuse one with the other. Despite their common appearance, XML and HTML technologies were created for different purposes. HTML is used for displaying data and XML is used for describing data.

Assume that you want to display, roses can be grown in California in spring. Its HTML representation would be:

```
<html>
      <body>
         <b><i>Roses can be grown in California during spring</i><b>
      </body>
</html>
```

Thus the tags (<b>, <i>) inform the browser how the data inside the tags (Roses…) should be presented. When the browser obtains the file from the web server, it has no idea what the information actually means. It displays the information inside the tags in bold and italic as indicated by the tags.

On the other hand, XML is a data representation language. It consists of node-value pairs. The tags represent the node name and the node value goes between the tags. The most important difference between XML and HTML are the tags. In HTML, tags are predetermined while in XML; tags can take on any value. An example XML document is:

```
<flowers>
   <flower>
      <name>Rose</name>
      <color>Yellow</color>
      <fragrance>Yes</fragrance>
      <growing season>Summer</growing season>
   </flower>
   <flower>
      <name>Daffodils</name>
      <color>Yellow</color>
      <fragrance>No</fragrance>
      <growing season>Summer</growing season>
   </flower>
</flowers>
```

This simple and minimalist document can educate the system a lot about the entity, flowers. It knows that flowers can occur in two types, roses and daffodils. It is also aware of the color, fragrance and growing season (various attributes) of the two flowers. The more exhaustive the XML document, the more knowledgeable the system becomes.

2.1.2 RDF and RDFS

Although XML can effectively be used to structure data, we can make it more powerful by adding new features. RDF [10] makes use of XML (it is an application of XML). It is used to express data in the form of classes and properties.

RDF uses web identifiers (URIs) to identify resources and describes resources with properties and property values. It uses the subject-predicate-object expression. Subject is the class, predicate is the property and object is the value.

An example of RDF would be:

```
<?xml version="1.0" ?>
<rdf:RDF
xlmns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xlmns:product = "http://www.shopping.com/prod#" >
<rdf:Description rdf:about="http://www.productdetails.et/Product1">
<product:description>Computer</product:description>
<product:price>500</product:price>
<product:quantity>1000</product:quantity>
<product:contains
rdf:resource="http://www.productdetails.et/Product2 />
      <ram>
```

```
            <rdf:Bag>
               <rdf:li>256</rdf:li>
               <rdf:li>512</rdf:li>
            </rdf:Bag>
      </ram>
</rdf:Description>
</rdf:RDF>
```

The <rdf:Description> element identifies a resource, Product1, with the about attribute. The properties description, price and quantity are defined in the http://www.shopping.com/prod# namespace and tell us more about resource Product1. The property, contains does not have a value but a reference to resource containing information about Product2, a product that Product1 includes. Product2 defines the RAM product and it comes in two sizes 256 MB and 512 MB.

RDF allows us to be more descriptive about data with tags like <rdf:Bag> (a set of possible values), <rdf:parseType="Collection"> (describe a group that contains only specified members) etc.

There is a need for RDF to define application-specific classes and properties. Application-specific classes and properties must be defined using extensions to RDF. One such extension is RDF schema (RDFS) [10].

Classes in RDF Schema are similar to classes in object oriented programming languages. Resources can be defined as classes, instances of classes, and subclasses of classes.

A RDFS example would be:

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base=  "http://www.birds.com/birds#">
<rdf:Description rdf:ID="bird">
<rdf:type
rdf:resource="http://www.w3.org/2000/01/rdfschema#Class"/>
</rdf:Description>

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

In the example above, the resource sparrow is a subclass of the resource bird.

2.1.3 OWL

OWL is defined by the W3C Web Ontology Working Group. OWL is a stronger language with greater machine interpretability than RDF. It also comes with a larger vocabulary and stronger syntax than RDF.

OWL can be used for creating and maintaining ontologies and knowledge bases. Ontology is a description of concepts and relationships occurring in the real world. Ontologies are created for the purpose of knowledge sharing and reuse. If ontology is viewed as a set of class definitions, then the knowledge base is a collection of the instances of those classes. In other words, ontology is a data model that represents a domain (definitions and relations) and a knowledge base contains objects belonging to that domain.

An OWL document is used for processing information on the Web. Computers can interpret OWL documents without human intervention. The following examples from OWL Web Ontology Language Guide [11] demonstrate the fundamentals of the OWL language.

Lets start building an ontology by defining OWL classes, `Wine` and `Winery`.

```
<owl:Class rdf:ID="Wine"/>
<owl:Class rdf:ID="Winery"/>
```

One of the main advantages of using ontology is that we can organize classes hierarchically allowing entire class definitions to be inherited.

```
<owl:Class rdf:ID="Wine"/>
   <rdfs:subClassOf rdf:resource="#Liquid" />
</owl:Class>
```

Thus an instance of `Wine` will always be an instance of `Liquid`.

Definitions may be incremental and distributed. In the sense, we can define the classes at the beginning of the document and state the class properties at the end of the document. An example property is given below:

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
   <rdfs:domain rdf:resource="#Wine"/>
   <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty"/>
```

The property `madeFromGrape` belongs to the `Wine` class and it takes on a value, which is an instance of the `WineGrape` class. Thus we are relating `Wine` and `WineGrape` classes using the property `madeFromGrape`.

We can impose restrictions on these properties.

```
<owl:Class rdf:ID="Wine">
<rdfs:subClassOf>
   <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:allValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
 </rdfs:subClassOf>
   .....
</owl:Class>
```

We are defining an unnamed class whose `hasMaker` property has to take on values that are members from the `Winery` class. Including this restriction in the `Wine` class definition body states that things that are wines are also members of this unnamed class. That is, every member wine should have an instance of `Winery` as its maker.

OWL has three expressive sublanguages. OWL Lite supports those users primarily needing a classification hierarchy and simple constraint features. OWL DL and OWL FULL have more expressive power and thus complexity. OWL DL is an extension of OWL Lite and OWL Full is an extension of OWL DL.

One of the features of OWL DL and OWL Full that is not available in OWL Lite is the availability of complex classes. A complex class is constructed as follows:

```
<owl:Class rdf:ID="WhiteWine">
<owl:intersectionOf rdf:parseType="Collection">
         <owl:Class rdf:about="#Wine" />
         <owl:Restriction>
            <owl:onProperty rdf:resource="#hasColor" />
            <owl:hasValue rdf:resource="#White" />
         </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>
```

The construction above states that `WhiteWine` is exactly the intersection of the class `Wine` and the set of things that are `white` in `color`. This means that if something is white and a wine, then it is an instance of `WhiteWine`. Without such a definition we can know that white wines are wines and that they are white, but not vice-versa. This is an important tool for categorizing individual entities. The `rdf:parseType="Collection"` is a required syntactic element.

The knowledge base is built by defining members belonging to the ontology classes. For example, members of `WhiteWine` (instances of the `WhiteWine` class) can be defined as follows:

```
<WhiteWine rdf:ID="StGenevieveTexasWhite">
   <locatedIn rdf:resource="#CentralTexasRegion" />
   <hasMaker rdf:resource="#StGenevieve" />
   <hasSugar rdf:resource="#Dry" />
   <hasFlavor rdf:resource="#Moderate" />
</WhiteWine>
```

Thus, the OWL ontology allows us to capture semantics of data allowing a web agent to access and process these machine-readable descriptions of the web content.

A variety of third-party tools that are described later can reason about the classes as well as their instances and infer information that is not explicitly stated.

Ontologies play a crucial role in the success of the Semantic Web. The Semantic Web envisions a Web wherein machines are capable of analyzing available data. This would involve building ontologies, knowledge bases and intelligent tools that are capable of processing them.

2.2 Semantic Web projects

Semantic Web projects can be broadly classified based on the goal they are trying to achieve:

1. infrastructure projects to support or act as a backbone for the Semantic Web,
2. projects to extract knowledge and construct an ontology and knowledge base and

3. projects to demonstrate the applications (use) of ontologies and knowledge bases.

2.2.1 Infrastructure

2.2.1.1 3 – Store

3– Store [12] is an implementation of a database server. The server is capable of storing huge amounts of RDF data. It consists of a library of C functions that interacts with MySQL to query, update and retrieve the knowledge base.

2.2.1.2 AKT-Bus

AKT-Bus [13] is defined as a low level (data-level) transport protocol. It's goal is to provide an infrastructure for distributed knowledge management and integration in a heterogeneous environment through a layered architecture and a message based protocol.

Some of the features of the AKT-Bus are use of HTTP for transportation, fine-grained XML for encoding the information to be transported, and a communication model between client and server in the form of request-response pairs.

2.2.2 Knowledge Extraction

2.2.2.1 Cockatoo

In Cockatoo, experts talk about the tasks they carried out resulting in knowledge sharing and construction. For example, task-oriented interviews were carried out with experts from the oil industry. The experts had to describe the rock structures, which they had investigated. Rock structures are composed of layers of rocks. Layers of rocks are specified by the rock type, hardness and rock thickness. The information they were providing had a definitive structure and hence a knowledge acquisition tool was written to extract this information from the expert directly.

Cockatoo attempts to perform grammar-driven knowledge elicitation. An example of the grammar and its representation from the Cockatoo fact-file [14] is given below.

Grammar:

```
formation -> <lithology>+
lithology -> (<rock> <lithology-depth> [<lithology-length>])
rock -> (<rock-type> <rock-hardness>)
rock-type -> (shale | clay | chalk | granite | other)
rock-hardness -> (very-soft | soft | medium | hard | very-hard)
```

It's representation in Cockatoo:

```
(defclause formation ::= (repeat+ <lithology>))
(defclause lithology ::= (seq <rock> <lithology-depth> (optional
<lithology-length>)))
(defclause rock ::= (seq <rock-type> <rock-hardness>)
(defclause rock-type ::= (one-of 'shale 'clay 'chalk 'granite
'other))
(defclause rock-hardness ::= (one-of 'very-soft 'soft 'medium 'hard
'very-hard))
```

Just as a XML schema validates a XML document, the above grammar will validate the knowledge extracted.

The grammar needs to be generated manually, which will then guide the knowledge extraction process. To help the user understand what information is required, each clause of a grammar is followed with a question and/or a comment. A question such as 'What is the rock-hardness?' is directed at the user. A comment provides additional information or other explanatory material. The information provided by the user will be validated by the grammar and the constraints within it.

2.2.3 Construction of Ontology and Knowledge Base

2.2.3.1 ANNIE

General Architecture for Text Engineering (GATE) [15] is an infrastructure for developing software components that process human language.

ANNIE builds on GATE and can recognize person names, organizations, locations, money amounts, dates, percentages, and some types of addresses in text.

The main language processing tools in Annie has been described in ANNIE - Open Source Information Extraction fact-file [16] and an excerpt of it is given below:

1. The tokenizer splits text into simple tokens, such as numbers, punctuation, symbols, and words of different types (e.g. with an initial capital, all upper case, etc.).
2. The sentence splitter segments the text into sentences. This module is required for the tagger. Both the splitter and tagger are domain and application-independent.
3. The tagger is a modified version of the Brill tagger, which produces a part-of-speech tag as an annotation on each word or symbol.
4. The named entity recognizer consists of pattern-action rules, executed by the finite-state transduction mechanism. It recognizes entities like person names, organizations, locations, money amounts, dates, percentages, and some types of addresses.

The online demonstration is available at Annie Demo [17]. It takes a URL as input. Example: With an article of George Washington [18] as input, it can recognize 'George Washington' as a person and 'February 22, 1732' as a date among several other things.

2.2.3.2 ADAPTIVA

The success of Semantic Web depends on the creation of a large number of domain specific ontologies. Many users may not know how to construct an ontology. Existing methodologies require a lot of expertise to construct an ontology. Manual ontology creation is slow and error-prone.

Ontology construction consists of creating a concept hierarchy, creating relationships between concepts and associating content with the concepts. It is possible to recognize terms and show that two terms are related in some manner using GATE and ANNIE. The next challenge would be how to label the nature of the relationship between the concepts as an ontology with explicit relations would be useful for problem solving [37].

Adaptiva [38] learns about relationships through a set of rules. Consider the relationship, ISA. A set of positive and negative examples is fed into Adaptiva.

ISA Relationship:

Example	Positive Example	Negative Example
fruits such as apples and oranges	Yes	No
…	…	…

For each positive example the algorithm builds an initial rule, generalizes the rule (to satisfy more number of examples) and retains the k best generalizations of the initial rule.

Once Adaptiva learns the rules, it is tested on an unseen set of examples. This iterative process may continue until a high proportion of examples are correctly classified automatically by the system.

Thus, Adaptiva uses a user-system interaction for ontology building.

2.2.3.3 Amilcare

Manual annotation is a difficult, slow and an error-prone process. Amilcare [19] allows automatic XML annotation of documents.

Amilcare is an extension of Adaptiva. Once Adaptiva has learnt rules about labeling relationships, Amilcare learns rules about document annotation for the Semantic Web through a set of positive and negative examples.

The basis of both Amilcare and Adpativa is the same: construct ontology and a XML knowledge base automatically using a set of rules. Rules are learnt using a set of positive and negative examples.

2.2.3.4 WebOnto

WebOnto's [20] aim is to develop a web-based ontology editor. It allows the user to browse, create and edit ontologies through the graphical interface. Although the ultimate goal would be automatic construction of ontology, till that is achieved, WebOnto provides easy user interfaces to do the same.

The user interacts with a graphical interface. Changes made by the user through the graphical interface are translated to actual ontology changes (in OCML language) by the code.

2.2.3.5 Dome

Dome [21] is a programming language for manipulating XML documents visually. It has a programmable XML editor, which is used to extract knowledge and transform web pages into RDF. The knowledge base is constructed from data coming in from many web pages. Since web pages keep changing, data in the knowledge base needs to be continuously updated. A tool is needed to do this automatically because of the large number of pages.

If all the pages provide machine-readable metadata, this task would be easier. But since most web pages are currently written in HTML, DOME can be used to build the knowledge base.

The web page to be processed is loaded into the DOME editor. User extracts knowledge from the page and builds an OWL document using Dome's user interface tools and programming constructs. The user actions are recorded in a program. When the web page content is changed, the user actions are replayed by Dome. The updated knowledge gets extracted automatically and the knowledge base is updated. Although Dome can cope with changes in the web page content, it may require human intervention to cope with web format changes.

2.2.3.6 ReTAX+.

ReTAX+ [36] identifies rules for building a consistent ontology. Some of the rules for building an ontology consisting of nodes and relationship between nodes are [36]:

1. each node must be more specialized than its parent,
2. each of a node's siblings must be unique,
3. is node N, already in existing hierarchy, T?,
4. to which parent node P, can N be attached and
5. to make N a child of P, do we need to make changes to N or T?.

Thus the project makes decisions on where to insert N based on the above rules.

2.2.3.7 Nmarkup

Nmarkup [22] is a web-based tool for building ontology from text. The tool is not completely automated and requires human intervention.

The tool highlights all the nouns in the text using Gate. The engineer uses NMARKUP to select concepts (using the highlighted nouns), relationships, attributes and values. The knowledge engineer can also add concepts, which are not in the text. After the engineer completes his selection, the tool will create a family of ontologies and display them graphically and in RDF. Further work to be done in the NMARKUP project includes recognizing and highlighting verbs that can act as relations and adjectives that can serve as attribute values.

2.2.4 Applications of an Ontology

The most important application of ontology is to build a knowledge base that can effectively represent real-world data. Real-world data consists of complex relationships between entities that map well into the Semantic Web. Following are some of the knowledge bases that have been successfully built:

2.2.4.1 AKT Research Map

The interesting feature in AKT Research Map [23] is the ontology design, which relates the various research activities. Its goal is to provide a high-level overview of how

the different researchers, tools and projects are collaborated and inter-related. The portal consists of two ontologies:

1. Support Ontology [24]: Defines generic classes like Thing, Tangible Thing, and Temporal Thing etc. These classes are not specific to the application.
2. Portal Ontology [25]: It consists of classes that are specific to the application. An example class would be publication, which will contain properties like name of author, publication date, publication place etc. A hierarchy of classes is built.

The AKT Research Map is composed of many individual maps. An individual map normally describes one researcher's activity. Individual maps consist of nodes (instances of ontology classes) and relationships between the nodes. Different maps share the same nodes. The individual maps are interconnected and they represent the AKT research activities. This design is similar to Entity Relationship modeling.

2.2.4.2 Eservices

The purpose of Eservices [26] is to semantically relate a large number of publications. The main feature of the framework is that it provides extensive citation linking which allows faster browsing. Authors and their publications form the entity nodes. Publications are linked together using citations. Since it is a semantic representation, we can query the database in a number of ways: Highest publishing researcher, most significant papers (based on citation impact) etc.

2.2.4.3 MyPlanet

MyPlanet [27] is an ontology driven personalized web based service. Personalized newsletters are delivered to the readers based on their reading interests. MyPlanet has a collection of stories stored in its database. An ontology is built and a hierarchy of topics on which MyPlanet has stories on, is created. The user can select the topics that interest him. The advantage of this ontology driven data model over a basic keyword search is that say the user selects ‘Genetic Algorithms’ as an interesting topic. Now MyPlanet not only returns stories which deal with ‘Genetic Algorithms’ but also stories about ‘Projects’ that have research area as ‘Genetic Algorithms’ (as potentially interesting topics) and also stories about ‘People’ who have worked on ‘Projects’ with ‘Genetic Algorithms’ as the research area (as contact people for information on that topic). As the various topics are inter-related in the ontology model, inferences can be drawn about the reader’s interest and the newsletter can be personalized.

2.2.5 Other projects

2.2.5.1 Refiner++

Refiner++ [35] uses a classification algorithm and builds rules based on the training data. Let us assume we have a set of cases presented to Refiner++ that are classified into categories by the domain expert. Refiner++ infers a description for each of the categories. It reports the inconsistencies that exist in the data set, and suggests strategies to reduce the number of inconsistencies. If background knowledge is available, then it is used by Refiner++ to produce more clear descriptions. Given below are 2 examples that show how Refiner++ reasons about the data sets.

Example 1: Consistent data set

Case	Field 1	Field 2	Category
1	A	Red	1
2	B	Red	2
3	A	Blue	1
4	B	Blue	2

Category Descriptions
Category 1: field 1: A, field 2: {Red, Blue}
Category 2: field 1: B, field 2: {Red, Blue}
Discriminatory field is field 1.

Example 2: Inconsistent data set

Case	Field 1	Field 2	Category
1	B	Red	1
2	B	Red	2
3	A	Blue	1
4	B	Blue	2

Category Descriptions
Category 1: field 1: A or B, field 2: {Red, Blue}
Category 2: field 1: B, field 2: {Red, Blue}
No discriminatory field is present. Refiner++ will detect the inconsistency and suggest changes to the user to make the data set consistent.

2.3 Study of tools for building ontology based applications

2.3.1 Jena

Jena [28] is a Java framework for building Semantic Web applications. It provides a programmatic environment for the various semantic web technologies like RDF, RDFS and OWL. It also supports SPARQL, a query language for RDF and includes a rule-based inference engine.

2.3.2 Pellet

Pellet [29] is an open-source Java based OWL DL reasoner. Pellet API provides functionalities like species validation, checking consistency of ontologies, classifying the taxonomy and answering a subset of RDF queries in RDQL, a query language for RDF. Pellet can be accessed from Java applications using APIs that support Jena and OWL. An online demo on Pellet is available [30].

2.3.3 IBM Integrated Ontology Development Toolkit

The IBM toolkit [31] is an ontology toolkit for storing, manipulating, querying and inferencing of ontologies and their corresponding instances. This toolkit includes:

1. Eclipse-based environment for ontology building, management, and visualization,
2. a storage, inference, and query system based on relational database for OWL ontologies and
3. Ontology Definition Metamodel available as an Eclipse open-source project.

2.3.4 WebOnto

WebOnto [34] is a web-based tool that allows a user to visualize, browse and edit ontologies and knowledge models specified in OCML (a modeling language). There are a few differences between OWL and OCML. OWL is a description language and OCML is an operational modeling language. For example in OCML, properties are defined within the definitions of classes; in OWL, properties are defined independent of classes (the domain of a property can be the intersection of two classes). OCML also allows the specification of functions, procedures, rules and relations, while OWL does not.

2.3.5 Protégé

Protégé [33] is a free, open source ontology editor and knowledge base framework. Protégé implements knowledge-modeling structures and actions that support the creation, visualization and manipulation of ontologies written in various languages including OWL and for entering data. Protégé also supports a Java-based API for building knowledge-based tools and applications.

2.3.6 KAON

KAON is based on OWL-DL. The latest version of KAON, KAON2 [32] provides an API for programmatic management of ontologies written in various languages including OWL-DL. It includes an inference engine for answering SPARQL queries. It can be interfaced with Protégé and relational databases.

KAON2 currently cannot handle nominals. If ontology contains an ‘owl:one of’ class or an ‘owl:hasValue’ restriction, it will throw an error. Pellet, on the other hand, can handle nominals.

Chapter 3

Analysis of Semantic Extensions to Wikipedia

Semantic Extensions to Wikipedia builds on the already existing successful portal called Wikipedia [42]. In Wikipedia, the knowledge base is built by allowing users to enter information on a variety of topics. In Semantic Extensions to Wikipedia, we want to enhance that process through the addition of a constraint that enforces the conformance of Wiki knowledge bases to ontologies. That modification allows the knowledge base to be directly processed by machines without human intervention.

The thesis makes an assumption that, there exists a repository of ontology classes that can be searched for and accessed by users. Efforts are already under way to implement such a repository. The Ontolingua Server [39] and SchemaWeb [40] allow users access to shared reusable ontologies. Applications can import ontology classes to specify data. Software agents that wish to obtain real-time schema information while processing RDF data can also use ontologies.

Once the ontologies are made available, we face a problem of building large knowledge bases that conform to these ontologies. With the advent of WYSIWYG editors, users don't even need to know HTML to create content for the Web. It may not be fair to expect laymen to learn XML, RDF and OWL to create Web documents. Offering convenient tools is critical for the success of the Semantic Web.

One approach to solving the problem of creating such "well-formed" knowledge bases is to employ Machine Learning methods to tag documents based on ontologies. MnM [41] is an ontology based annotation tool for annotating web pages with semantic contents. It may be possible to train MnM in a limited domain wherein every ontology class comes with its set of rules to identify its attributes and relations. However as rules are specific to an ontology class and performance is sensitive to the quality of training data, to prepare the learning corpus and perform training over several thousand ontology classes is an arduous and time-consuming task.

The other approach, taken up by this thesis is to carry out a dialog with the user, based on the ontology and retrieve information in a format suitable for building a knowledge base. The ontology maps naturally occurring entities to well defined concepts. For example, the entity 'Fruit' is defined by its color, shape, size and edibility. This definition can be expressed in the ontology using OWL. When the user wishes to enter information on 'Banana' (an instance of Fruit) he is asked questions like 'What is the color of Banana?', 'Is the Banana edible?' etc. The user answers the questions and the extracted information goes back to the knowledge base. This question-answer module can lead to transparency in building knowledge bases. The knowledge base is structured in SEW

unlike in Wikipedia. A user may forget to enter important facts while submitting his articles to Wikipedia. In SEW, information will be complete if the user answers all the questions and if the quality of the supporting ontologies is ensured. Like Wikipedia, information from the knowledge base can be retrieved and updated in SEW.

3.1 Use Case Diagram

Given below is the use case diagram, which shows how SEW is going to be used. The diagram identifies the actor (person interacting with the system), use cases (sequences of actions that can be performed by the actor on the system), and the associations between the two.

In Wikipedia, there are different types of users: the editors (everyone from the expert scholars to the casual readers), the administrators with special powers to enforce good behavior and the members of judicial committee that can take action (withdraw or restrict editing privileges) in an effort to control vandalism.

In SEW, we identify a single type of user: the editor who visits the website to retrieve and update information. Other users with different roles can be added on in the later versions of the system. The editor can interact with the system in a number of ways such as

1. uploading the ontology onto the server,
2. browsing through available ontologies and
3. selecting ontology and retrieving or editing the corresponding knowledge bases.

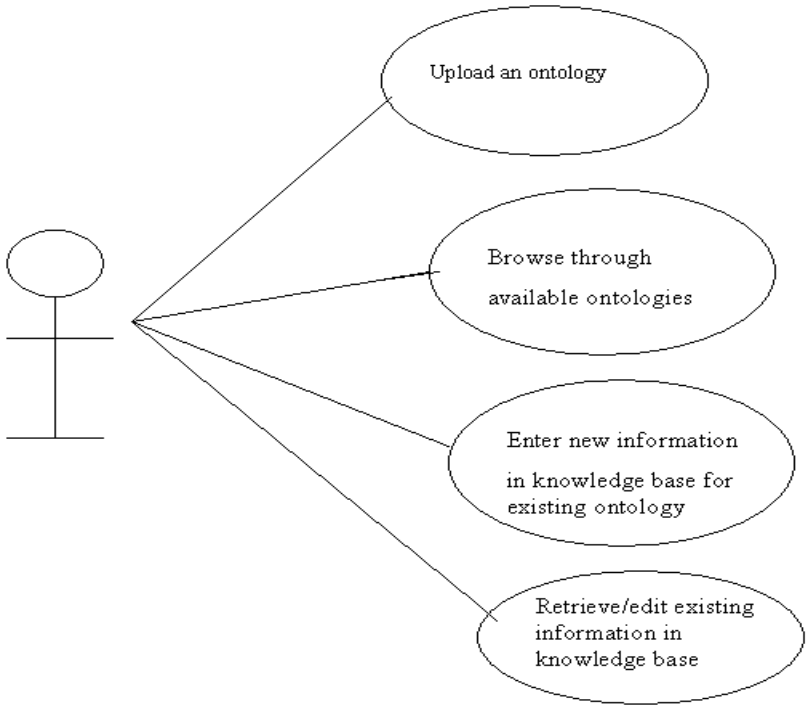


Figure 3.1: SEW Use Case Diagram

3.2 System Architecture

Let us assume a scenario when the client/user submits a HTTP request to the server; to retrieve information. The server does double duty: it acts as both a web server (capable of handling HTTP requests from the browser) and application server (capable of executing Java applications). The server intercepts the request and maps the URL to an appropriate handler, in this case, a servlet. A servlet is a Java program that resides on a server and can access the server resources. The servlet processes the input and invokes the application component residing on the server capable of generating the required response. It could either be a RDFReactor program that contains the business logic of converting OWL classes to Java classes, or the Database Manager, which interacts with the knowledge base. In the latter case, the server calls the Database Manager that retrieves the desired information from the knowledge base. The information and program control is returned to the calling servlet. The servlet calls upon the appropriate Java Server Page (JSP). JSP is a technology that renders information dynamically in a HTML document. The server sends the HTTP response back to the client and the user is able to see the retrieved information in his browser window.

The architecture follows the MVC (model view controller) design pattern. The view component or JSP pages are responsible for presenting data to the user. The controller component - or servlet - intercepts the requests and routes the responses between the view and model components. The model component (that is the RDFReactor, Database Manager, Ontology Store and Knowledge Base) contains the domain logic and data for input processing and output generation.

The following high-level diagram illustrates the components of Semantic Extensions to Wikipedia and the relationships between them.

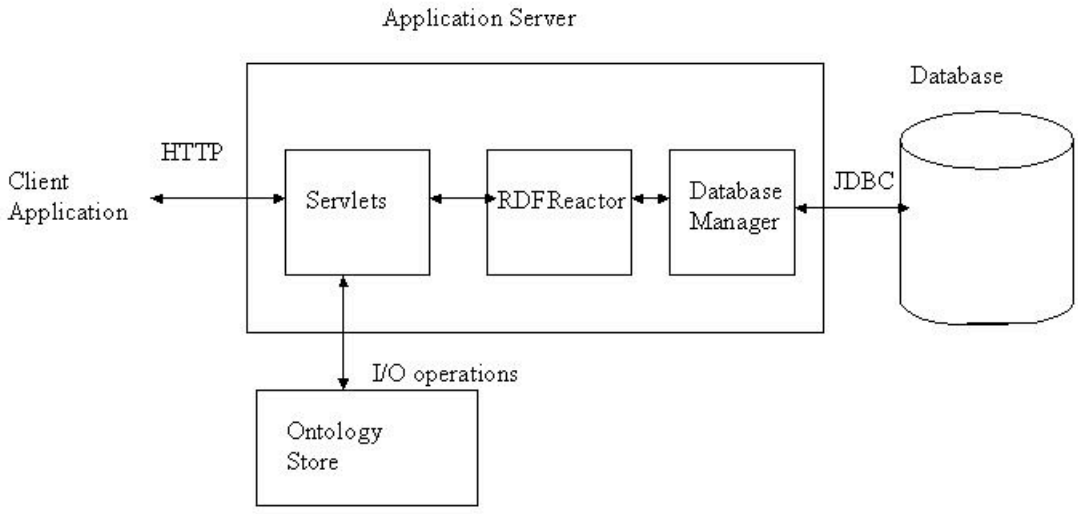


Figure 3.2: SEW System Architecture Diagram

3.3 Process Flow

In this section, we will discuss various process flows in Semantic Extensions to Wikipedia.

3.3.1 User uploads an Ontology

In this thesis, we try to map naturally occurring entities to well defined concepts. Concepts can be related to each other, inherited and extended to model the real world. Ontology is made up of concepts and it forms the underlying data model for the knowledge base we are trying to build. Automated machine processing of data can be achieved when machines can look up the meaning of information in the ontology. Complete and expressive ontologies will lead the way to superior data reasoning.

SEW attempts to build a library of ontologies by allowing users to upload ontologies. Users can create their own ontologies using tools like Protégé or download them from a directory like SchemaWeb. SEW is not concerned with the origin of the ontology provided that it conforms to the OWL specifications released by the W3C.

After the ontology is uploaded, it needs to be processed. There are several tools available for manipulating OWL ontologies. We chose RDFReactor because it can convert an OWL ontology into Java classes. RDFReactor acts as a bridge between the Java web application and the OWL data model. The mapping from the OWL schema to Java is automated and customizable. Also, all the great features of Java are available for extracting definitions and manipulating the ontologies after the conversion.

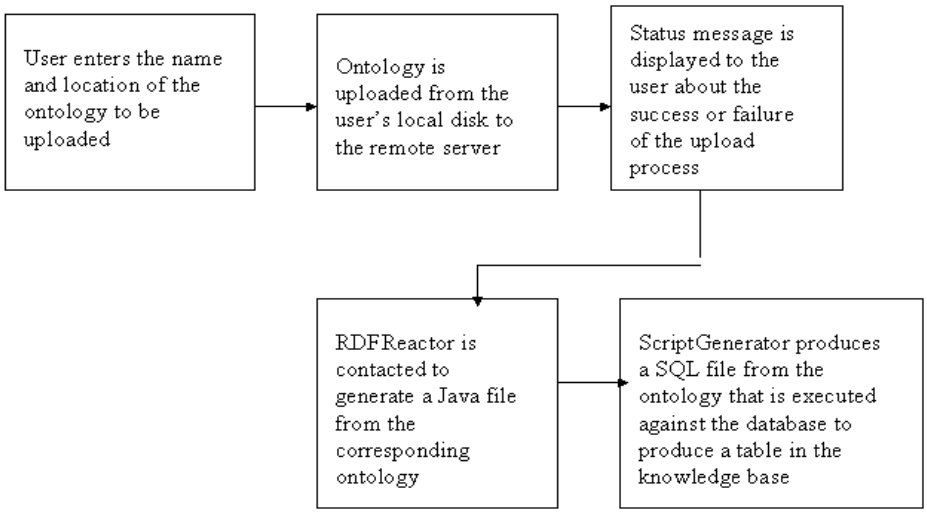


Figure 3.3: Diagrammatic Representation of process 'User uploads ontology'.

The structure of the knowledge base is created from the ontology. The reason being that, we could then extract and store information from the user in a predetermined format that would facilitate later processing. We decided to go with the relational database to store the knowledge. By storing the knowledge in database we are preserving the knowledge structure through the table definitions. As the database is relational, relations between the ontology classes can also be preserved. Java classes would map to tables in the relational database. Class properties would form the table column names and property

types would be mapped to column types. There would also be a mapping of relations and constraints between the Java classes and the relational database tables. The ScriptGenerator module is responsible for creating the database from the ontology.

3.3.2 User browses through existing ontologies

A knowledge base is dependent on its ontology. The ontology informs the machine what the information in the knowledge base means and how to process it. So in SEW, we require that the user choose the ontology before letting him access the associated knowledge bases. This could be implemented either by a 'Search' or 'Browse' functionality; we chose the latter. The existing ontologies are retrieved from the remote server and displayed to the user. The user can select the ontology whose knowledge base he wishes to access.

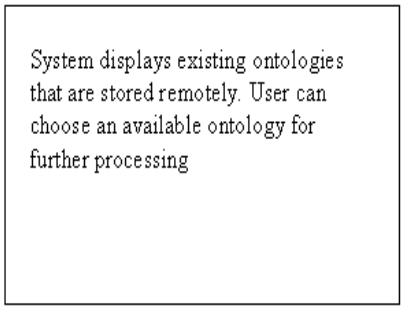


Figure 3.4: Diagrammatic Representation of process 'User browses through existing ontologies'

3.3.3 User wishes to enter a new record in the knowledge base for a given ontology

After selecting the ontology, the user can retrieve information from the knowledge base; edit existing information or input new information. If the user wants to key in new information there are 2 ways of implementing this:

1. allow the user to enter a text document, extract information from the document into the knowledge base using the ontology and
2. ask questions to the user using the ontology and extract the information from the answers into the knowledge base.

We went with the second approach, as the first approach requires a lot of natural language processing which is not in the scope of this project. Let us assume that the user wishes to enter information about 'oranges'. Using the fruit ontology, we do know what data to expect from the user and can prompt him about the same. Assume that the ontology defines a fruit as an entity having color, taste, size and edibility properties. Therefore, the machine starts a conversation with the user, 'What is the color of oranges?', 'Are oranges edible?', and so on.

The advantage of this approach is that it is user friendly. The user does not need to prepare a whole document on a topic. He just answers the questions posed by the system and signs off. This straightforward approach may encourage him to contribute more, maybe key in information on other fruits like 'apples' and 'bananas'.

The process flow diagram is given below. First, the user selects an ontology. Questions are generated dynamically from the definitions in the ontology. When the questions are answered, the knowledge base is updated. As the knowledge base grows, the repository of machine-processable information increases. The information is reusable and can be shared by exposing it to other applications through suitable interfaces.

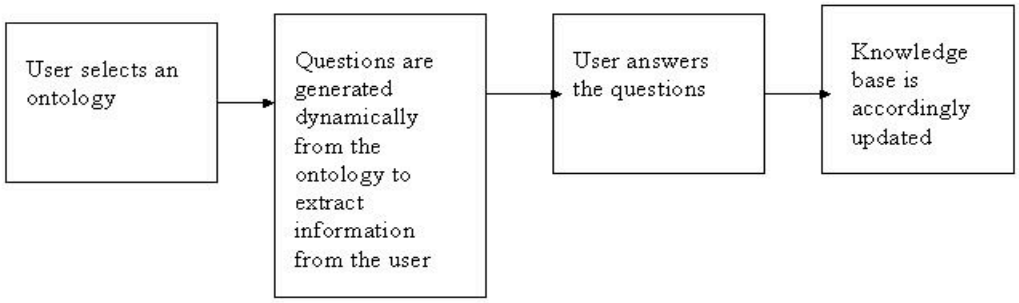


Figure 3.5: Diagrammatic Representation of process 'User enters a new record'

3.3.4 User wishes to edit the knowledge base

When the user selects the ontology, he can retrieve or update information in the knowledge base. Let us assume that the user wishes to edit existing data.

Firstly, all the data stored in the knowledge base for that ontology is displayed to him in a tabular format. This step can also be viewed as the 'Information Retrieval' functionality in SEW.

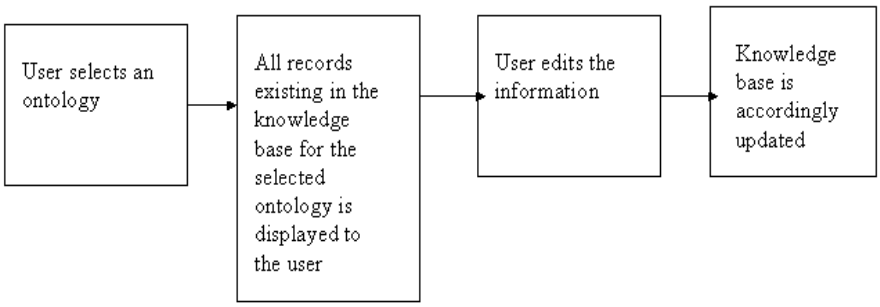


Figure 3.6: Diagrammatic Representation of process 'User edits knowledge base'

Several simultaneous updates are possible. User can edit the information present in the records displayed and the updated data goes right back to the knowledge base.

Chapter 4

Implementation of the SEW

In this chapter, we will list the various software and tools used in the project. For each of them, we discuss the installation and the reasons why we chose the particular software or tool and how it is used in the project. We will also discuss the problems that we faced in the installation, use and interfacing of the software/tool with our application and the problem resolutions.

Next, we do a code walkthrough. We examine the flow of control from the front end (JSPs and servlets) through the middle tier (application files containing the business logic) to the back end (DBManager containing code to interact with the database). Important code snippets are also included with their supporting explanations.

4.1 Examination of software tools used in the project

4.1.1 Eclipse

Eclipse [43] is an extensible and open source integrated development environment. The Eclipse platform, when combined with the Java Development Tools (JDT), offers many features: a syntax-highlighting editor, incremental code compilation, source-level debugger, a class navigator, a file/project manager and interfaces to version control systems like CVS [44].

The primary requirement for the Eclipse installation is a Java Virtual Machine and the version we are using is 1.5. After installing Eclipse, we specify the workspace, the directory on the hard drive where Eclipse stores the projects that we define. Next we create the Java Projects, which involves the creation of packages and source and class folders within the packages. Use of packages allows us to organize files based on functionality.

We also installed the JdtAptFeature from the Eclipse Update Site for annotation processing.

4.1.2 MySQL

MySQL [45] is an open source relational database system. After installing MySQL, we configure server instance and set the password for access into the database. The Java program executes commands against the database using the Java Database Connectivity

(JDBC) API. MySQL Connector/J (a JAR file) is the official JDBC driver for MySQL and it is stored in the lib directory of the Tomcat server.

We connect to `port 3306` on the default secure database `mysql`. SQL commands are generated dynamically by the Java programs and executed.

4.1.3 Tomcat

An application server [47] is a server computer that is installed in a computer network. It accepts requests from client computers, executes software applications and delivers responses if required back to the client computers. In some cases, client computers could be dummy terminals that execute applications on the server remotely. Application servers also offer features like security and transaction processing. For a huge application serving thousands of requests, we may need to deal with issues like scalability and fault tolerance while using an application server.

The web and application server used in this project is Tomcat [46]. First Tomcat is downloaded and unzipped at a location of our choice [48]. Then we set the `JAVA_HOME` environment variable to inform Tomcat where to find Java. The default port on which Tomcat runs is `8080` and this setting need not be changed. servlet.jar, which is required to process Java servlets and JSPs is downloaded into `TOMCAT_HOME/lib/common` directory. Server is started to verify that the installation is correct.

Next we need to create the development environment, the directory which hosts the servlets, JSPs, templates and class files. The `webapps` folder under `TOMCAT_HOME` hosts the various applications. We can create a folder for our application called `SEW` under the `webapps` folder.

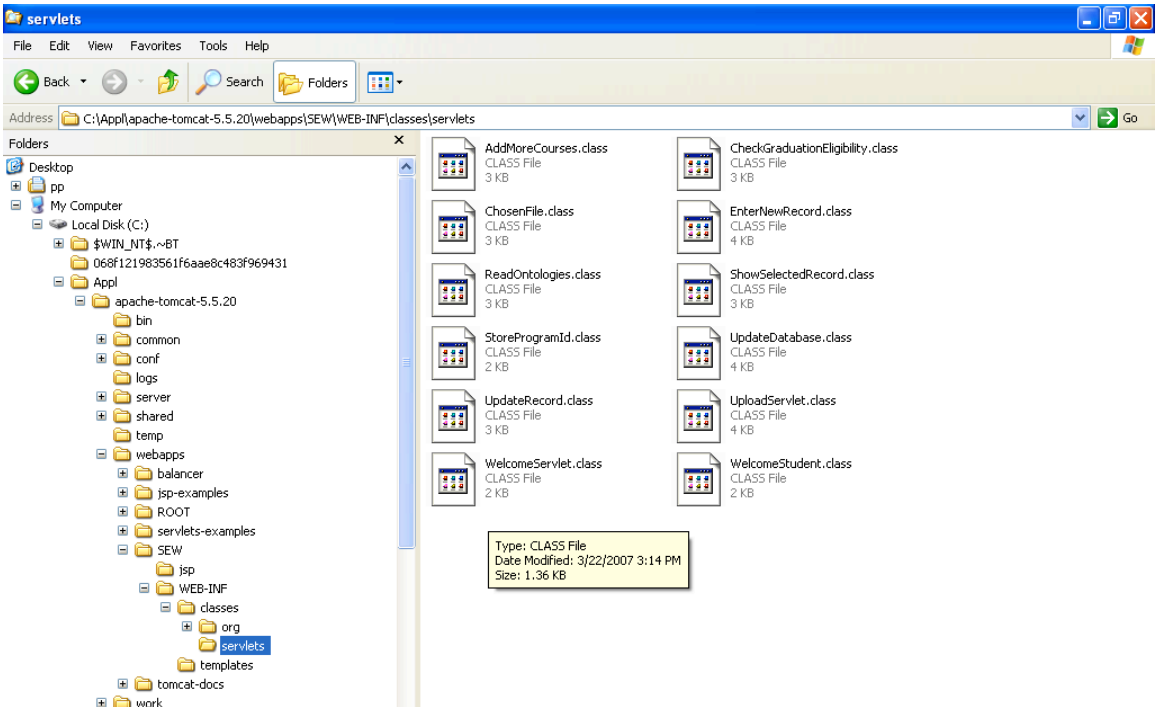


Figure 4.1: The screenshot of application deployment in Tomcat (tree view).

The folder structure can be explained as follows:

1. JSPs goes in `TOMCAT_HOME/webapps/SEW/jsp` folder,
2. web.xml is in `TOMCAT_HOME/webapps/SEW/WEB-INF` folder,
3. templates goes in `TOMCAT_HOME/webapps/SEW/WEB-INF/Templates` folder and
4. servlets can be found in `TOMCAT_HOME/webapps/SEW/WEB-INF/classes/servlets` folder and the package containing the application class files goes in `TOMCAT_HOME/webapps/SEW/WEB-INF/classes` folder.

After starting the server, we can visit our application home by typing the address `http://localhost:8080/SEW/jsp/welcome` in the browser window. This means we are using the HTTP protocol to communicate with the localhost (as the application is hosted on the local machine) through the default port 8080. SEW is the name of the application and welcome.jsp is found in the SEW/jsp folder and it is rendered in our browser window.

4.1.4 Java

The object oriented programming language used in the project is Java [49]. It is platform-independent (after writing and compiling a Java file, the byte code can be executed on diverse hardware) and its design allows the code to be executed from remote sources securely and transparently.

Java also supports reflection. When we write a program, we specify the sequence of operations that access and modify objects. When we compile the program, the class file is generated and the behavior of the program gets fixed. There will be no surprises during the execution of the program. However there will be cases where we want to add flexibility to the program execution based on the data object the program accesses. Let us assume we have a program `BuyItem.java`. In this program, if the item is a book we want to call `checkReviews()` method and if the item is a car we want to call the `checkGuarantee()` method before buying the item. The item to be bought will be known only when the user actually buys the item and not beforehand when we are writing the program. Thus we cannot specify the method name at compile time. Instead we need to wait till runtime, examine the object to be bought and then specify the method to be called. This kind of program behavior can be achieved through reflection.

Reflection is a property that allows a Java program to examine an object at run time. For an object `o`, we first do a `o.getClass()` to access the class c, represented by the object. Once we have a handle to c, we can access the object's constructors, super classes, fields and methods. This information is not available beforehand and is made available at run time through reflection. Reflection is widely used in the project and it will be discussed further in the code overview. The Java version we use in the project is JDK 1.5 because the RDFReactor module uses several features that are not available in the previous versions of Java.

4.1.5 RdfReactor

Semantic Extensions to Wikipedia is a Java based web application. The ontologies that are the backbone of the project are written in OWL, which is a part of the Semantic Web technology. To make the two technologies interoperable, we make use of RDFReactor [50]. RDFReactor is a tool that transforms a given ontology in RDFSchema/OWL into an object-oriented Java API. Developers can easily interact with the Java instances.

A RDF Schema (or an OWL ontology) is run through the RDFReactor [51] code generator, which creates type-safe, domain-specific Java classes. Instances of these classes act as stateless proxies on the RDF model at run-time. RDFReactor converts OWL classes to Java classes and RDF properties to Java properties, accessed through `getValue()` and `setValue()` methods. All API methods are type-safe. Hence a developer using a RDFReactor generated API has full support from the Java compiler and it's IDE, e.g. auto-completion of method names.

The Java classes and methods are named according to usual Java coding conventions. After code generation, the API is ready to use. It's fully implemented and documented. The Java classes can also handle inheritance, cardinality constraints, domains and ranges found in the OWL ontology. The Java classes are generated using a template engine, Velocity discussed below. The resulting classes can be compiled without errors and their structure is predetermined as they are generated from a template.

Although RDFReactor has a fully implemented abstraction layer, RDF2GO, to manipulate the RDF model through the generated API, we have written SEW specific Java programs to access and use the API. After writing some interfacing code, RDFReactor was plugged into SEW and was ready to use.

4.1.6 Velocity

Velocity [52] is a Java-based template engine. It permits a user to use a powerful template language to reference objects defined in Java code. It can also generate JSP, SQL, PostScript and XML from templates. In this project, Velocity is used to generate Java source code from a template, `class.vm` (available in RDFReactor) containing markup statements. These statements could be a class, property or method definition statements, basic control statements that can loop over a collection of values (`for each`) or conditionally show a block of text (`if/else`) etc. These statements contain placeholders whose values are inserted dynamically, in our case from the ontology.

An implementation problem that we faced was where to place the `class.vm`. In Eclipse, we could put it in the current directory and it would be accessed. However Tomcat was not able to find the template, even though we tried putting it in a number of locations. We did not want to include the complete absolute path because if the template path changes at a later date then it would be cumbersome to go through the entire source code and change all references. The resolution to this problem was: It didn't matter where we had the template so long as we included the template path in Tomcat `classpath`. Tomcat `classpath` is defined in the `setclasspath.bat` file in the `bin` directory. If the template path changes, we need to update only one file, `setclasspath.bat` where the classpath is defined.

Initially we wanted to generate the SQL scripts from the ontology using Velocity. However we did not find much documentation on it and decided to code it ourselves. We wrote SQL statements containing Java variables whose values came either from the user input or from the Java file created from the ontology. For example consider a part of the script, `create table tablename` where `create table` are the SQL keywords and `tablename` is the name of the Java file created by RDFReactor.

4.2 Code walkthrough

Consider a sample process: 'Upload an ontology'. To highlight the salient features of the code, we will see how the code is executed to process the request and return results.

To upload ontology, we need to specify the name of the file to be uploaded and it's location on the user's local disk. When the `Upload Ontology` button is pressed, a HTTP request is sent from the browser to the server in the form of a URL: http://localhost:8080/SEW/uploadservlet. Now the server tries to match the URL with the URL patterns listed in the `web.xml`. If an exact match is not found, the server will try to match the longest path prefix with '/' as the path separator. In this case, a matching URL pattern is found for the longest path prefix `uploadservlet`. The corresponding servlet [55], UploadServlet is then invoked. The `web.xml` contains entries as follows:

```
<servlet>
        <servlet-name>
            uploadservlet
        </servlet-name>
        <servlet-class>
            servlets.UploadServlet
        </servlet-class>
</servlet>

<servlet-mapping>
        <servlet-name>
            uploadservlet
        </servlet-name>
        <url-pattern>
            uploadservlet
        </url-pattern>
</servlet-mapping>
```

It contains 2 mappings for each servlet: a mapping of the servlet name with the class file, and mapping of the servlet name with the URL pattern. Now since our request's URL matches with URL pattern of `UploadServlet`, the corresponding class file is invoked.

Thus `web.xml` directs the request from the browser to the `UploadServlet`. `UploadServlet` reads in the request parameters, `File name and Location`. It processes the request by copying the file bytes from the user's local disk to a specified location in the server, in this case `D:/ontologies`. Thus the file is uploaded [56]. Next, we need to create a Java file from the OWL file. This conversion is taken care of by the RDF module. The servlet calls the `generateRDFReactor()` method in `Manual.java`, which is the entry point for the RDF module.

First, `wine.owl`, the uploaded file is read into the memory and it looks like this:

```
<rdf:RDF xmlns="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:w="http://www.w3.org/2006/10/wine#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

   <owl:Ontology rdf:about="">
      <rdfs:comment>An example OWL ontology</rdfs:comment>
      <rdfs:label>Wine Ontology</rdfs:label>
   </owl:Ontology>

   <owl:Class rdf:ID="Wine">
   </owl:Class>

   <owl:ObjectProperty
   rdf:about="http://www.w3.org/2006/10/wine#Name">
   <rdfs:domain rdf:resource="#Wine" />
   <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
   </owl:ObjectProperty>

   <owl:ObjectProperty
   rdf:about="http://www.w3.org/2006/10/wine#Color">
   <rdfs:domain rdf:resource="#Wine" />
   <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
   </owl:ObjectProperty>

   <owl:ObjectProperty
   rdf:about="http://www.w3.org/2006/10/wine#Flavor">
   <rdfs:domain rdf:resource="#Wine" />
   <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
   </owl:ObjectProperty>
</rdf:RDF>
```

In this file, we define a class called `Wine`. Then we define three properties, `name, color and flavor`. Their domain is defined as `Wine` and this means that the properties belong to the `Wine` class.

RDF reactor first scans the owl file to extract the classes. In `wine.owl`, we have a class called `Wine` and hence the program generates the corresponding Java file, `Wine.java`, with constructors that allow us to create instances of the class. Then `wine.owl` is scanned for properties whose domain is `Wine`. We have 3 such properties, `name, color and flavor` which now become properties in `Wine.java` and can be accessed by getter and setter methods. As OWL can allow a property to have multiple values, we have the `addValue, removeValue and getAllValue` in the Java file to accommodate multiple values.

Velocity engine in the RDF module creates the Java file using the template `class.vm`. Part of the template is shown below:

```
package $package.name;
public class $class.name extends $class.superclass {
//setter method
   public void set#mixedcase( $prop.name )( ${classname} value) {
      set(#uppercase($prop.name), value);
   }
}
```

The template contains placeholders that are filled in from the Jena model that the program has built by extracting values from the Owl file. The placeholders in the template and the content that replaces it in the Java file are highlighted in red. Part of the generated Java file is shown below:

```
package org.ontoware.rdfreactor.example;
public class Wine extends Thing {
//setter method
   public void setName( java.lang.String value ) {
      set(NAME, value);
   }
}
```

After generating the Java file, we generate the class file. The class file is generated programmatically. The Java compiler tool is available in the com.sun.tools.javac.Main package. We need to specify two parameters, the classpath which includes the location of the Java file and the output directory within Tomcat where the compiled class file should be generated. By invoking the compile method of the tool, the class file is generated at the specified output directory. All subsequent accesses will be made to the class file.

Each class will have multiple instances. The class file can be viewed as the metadata for the instances. In this project, we store the instances in the knowledge base implemented by a relational database. The class file will define the structure of the tables in the database.

Next, we read the class file into memory, generate a script from it dynamically and execute it against the database to produce the table. In this example, `Wine.class` is read into memory and we process it using reflection [53] as follows:

1. the variable `javaClass` represents the class file: `Wine.class`,
2. properties can be obtained using `javaClass.getDeclaredFields` and are stored in the variable `javaProperties`,
3. methods can be obtained using `javaClass.getMethod` and are stored in the variable `javaMethods` and
4. property names and data types are obtained using `javaProperties.getName()` and `javaMethods. getReturnType()`.

After getting the property names and return types, we can proceed with the `create.sql` script. The script skeleton is as follows:

```
create table tableName (
   fieldName1 dataType1,
   ……
   fieldName3,dataType3);
```

where:

1. tableName is `Wine` extracted from the class name and
2. field Names and datatypes are generated from the property names and method return types.

After creating the script, we pass it on to the `DBManager`. The `DBManager` establishes a connection with the database [54] as follows:

a. Register the JDBC driver `(com.mysql.jdbc.Driver)` for MySQL.
b. Define URL of database server `(jdbc:mysql://localhost:3306/mysql)` for database named `mysql` on the localhost with the default port number `3306`.
c. Get a connection to the database (`DriverManager.getConnection(url,"root", "rang50")`) for a user named `root` with the specified password. This user is the default administrator having full privileges to do anything.
d. The script can now be executed and the table is created in the database.

The process of uploading the ontology, dynamically generating the Java file using template and script using reflection is now complete. Control is returned back to the servlet. The servlet sends back a URL through the request dispatcher. web.xml matches the URL pattern with a JSP. The JSP is returned back to the browser, which renders a status message saying that the upload process was successful.

The highlight of this process is its transparency. The user just knows that his information has been correctly submitted and received. He is blissfully unaware of the back end processes triggered by his request.

Other processes in the project are executed as follows:

1. In the `Browse Ontology` process, the servlet accesses the directory on the server, which hosts the ontologies, obtains the names of files and displays it to the user.
2. In the `Enter a new record in the knowledge base for existing ontology` process, the servlet passes on the control to the middle tier along the existing ontology name chosen by the user, as the parameter. The middle tier looks up the class file (generated from the Java file) representing that ontology. Fields (properties) of the class are extracted using reflection. Questions are formed using the fields e.g. if the field extracted is `name`, then the question could be: `What is the name of the wine?` The questions are displayed to the user.
3. After the user answers the question, he will submit the information. The servlet will extract the data (field names and field values) from the request and pass it on to the middle tier. The middle tier will generate an `insert` script dynamically from the data. The script is passed on to the `DBManager`. On execution of the

script, a row will be created in the database. This row will contain the information submitted from the user. Thus unstructured information gets structured and is appended to the knowledge base.

4. In the `Retrieve and Edit records in knowledge base` process, user again chooses the ontology whose knowledge base he wishes to retrieve and edit. Existing information from the database is retrieved by the `DBManager` and displayed to the user through a `select` script generated dynamically by the middle tier. The displayed information is editable. The user can edit the data and submit it. The `UpdateDatabase` servlet extracts the data (modified field values), the middle tier constructs an `update` script using the field names and values and `DBManager` fires the script. Information in the knowledge base is thus updated.

Chapter 5

Experimenting with the SEW

In this chapter, we will test SEW to evaluate its performance in the real world domain. We will use SEW to check if a student has completed the coursework required for graduation. Student will enter the program he has enrolled into. He will also enter the courses he has taken. SEW will use the ontologies, knowledge bases and rules to process the input. SEW will output a message indicating whether the student has fulfilled the program requirements or not.

We will begin by building the ontologies to define the domain. We will create knowledge bases based on the ontologies. The data for the knowledge bases comes from the CSUCI MSCS website [59]. We will also write rules on how to process the input to get the desired output.

We will conclude this chapter by listing the observations from the experiment.

5.1 Building the ontologies

Each program in the university defines a curriculum. The curriculum lists the courses associated with the program. It also defines the requirements for graduating from the program.

5.1.1 The Course ontology

The first ontology we will build is the Course ontology. We have 3 types of courses: required, core and elective courses. The student needs to take all the required courses in the program. He has to take a certain number of core courses and elective courses.

We will define the Course object in the ontology using the attributes: id, name, number of credits and type. The definition is generalized and can be applied to different programs in the same university or different universities.

The Course ontology is as follows:

```
<rdf:RDF xmlns="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:w="http://www.w3.org/2006/10/course#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
```

```
    <owl:Ontology rdf:about="">
       <rdfs:comment>A course ontology</rdfs:comment>
       <rdfs:label>Course Ontology</rdfs:label>
    </owl:Ontology>

    <owl:Class rdf:ID="Course">
    </owl:Class>

    <owl:ObjectProperty
    rdf:about="http://www.w3.org/2006/10/course#Id">
       <rdfs:domain rdf:resource="#Course" />
       <rdfs:range
      rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
    </owl:ObjectProperty>
    <owl:ObjectProperty
    rdf:about="http://www.w3.org/2006/10/course#Name">
       <rdfs:domain rdf:resource="#Course" />
       <rdfs:range
      rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
    </owl:ObjectProperty>
    <owl:ObjectProperty
    rdf:about="http://www.w3.org/2006/10/course#Credits">
       <rdfs:domain rdf:resource="#Course" />
       <rdfs:range
       rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty
    rdf:about="http://www.w3.org/2006/10/course#Type">
       <rdfs:domain rdf:resource="#Course" />
       <rdfs:range
      rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
    </owl:ObjectProperty>

</rdf:RDF>
```

5.1.2 The Program ontology

The program will define the number the core courses and elective courses the student has to take. All the required courses need to be taken. Hence the number of required courses the student needs to take is not defined.

We will define the Program object in the ontology using the attributes: id, name, number of core courses and number of required courses. The definition is again generalized to promote reusability.

The Program ontology is as follows:

```
<rdf:RDF xmlns="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:w="http://www.w3.org/2006/10/program#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

   <owl:Ontology rdf:about="">
```

```
    <rdfs:comment>A program ontology</rdfs:comment>
    <rdfs:label>Program Ontology</rdfs:label>
  </owl:Ontology>

  <owl:Class rdf:ID="Program">
  </owl:Class>

  <owl:ObjectProperty
  rdf:about="http://www.w3.org/2006/10/program#Id">
    <rdfs:domain rdf:resource="#Program" />
    <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
  </owl:ObjectProperty>
  <owl:ObjectProperty
  rdf:about="http://www.w3.org/2006/10/program#Name">
    <rdfs:domain rdf:resource="#Program" />
    <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
  </owl:ObjectProperty>
  <owl:ObjectProperty
  rdf:about="http://www.w3.org/2006/10/program#CoreCourses">
    <rdfs:domain rdf:resource="#Program" />
    <rdfs:range
    rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty
  rdf:about="http://www.w3.org/2006/10/program#ElectiveCourses">
    <rdfs:domain rdf:resource="#Program" />
    <rdfs:range
    rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
  </owl:ObjectProperty>

</rdf:RDF>
```

5.1.3 Association Between Program And Course ontology

We need to associate courses with their respective programs. For example the course PHY 510 is common to both the Math and Computer Science graduate programs and hence needs to be associated with both the programs. A student can take a large number of courses, but if the courses are not included in the program curriculum, he may not be able to graduate.

We will define the AssociationBetweenProgramAndCourse object in the ontology using the program id and course id. The relationship between the attributes is many-many. A program id can be associated with multiple course ids and a single course can be associated with multiple programs.

The Association Between Program And Course is as follows:

```
<rdf:RDF xmlns="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:w="http://www.w3.org/2006/10/course#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
```

```
<owl:Class rdf:ID="AssociationBetweenProgramAndCourse">
</owl:Class>

<owl:ObjectProperty rdf:about="Number">
   <rdfs:domain
   rdf:resource="#AssociationBetweenProgramAndCourse" />
   <rdfs:range
  rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="ProgramId">
   <rdfs:domain
   rdf:resource="#AssociationBetweenProgramAndCourse" />
   <rdfs:range rdf:resource="#Program:Id" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="CourseId">
   <rdfs:domain
   rdf:resource="#AssociationBetweenProgramAndCourse" />
   <rdfs:range rdf:resource="#Course:Id" />
</owl:ObjectProperty>
</rdf:RDF>
```

5.2 Uploading the ontologies

When the user logs into the system, the welcome page greets him as follows:

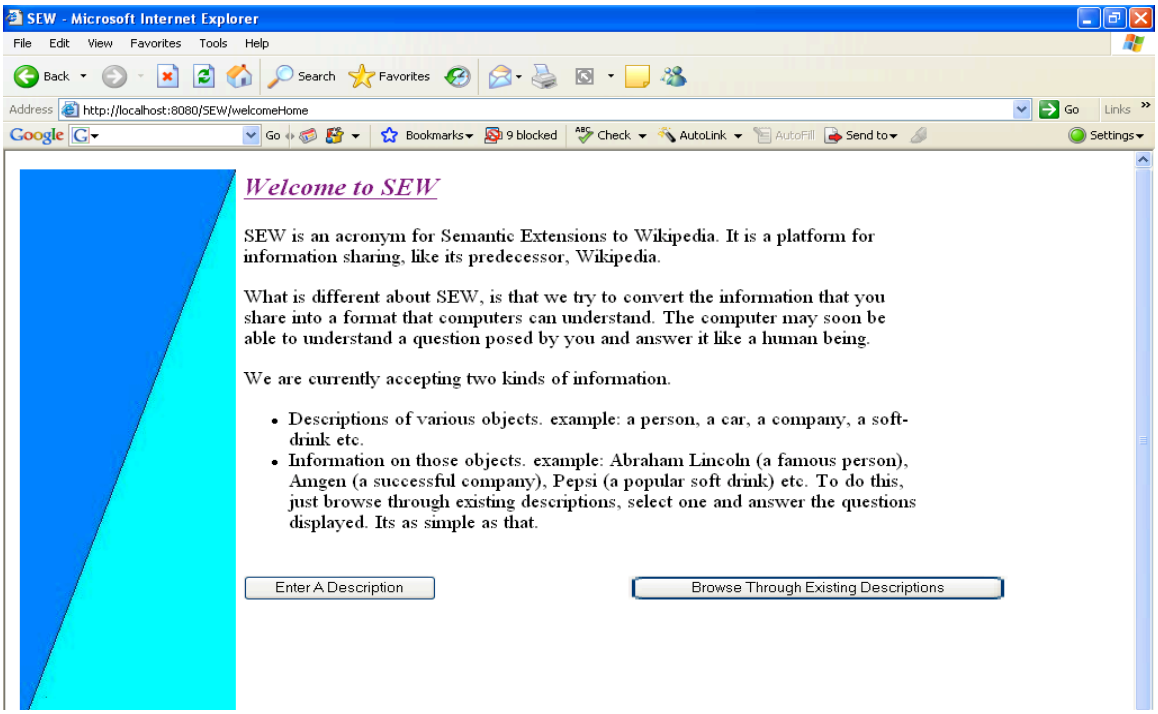


Figure 5.1 Screenshot of the Welcome page

The user clicks on `Enter a Description` button.

The user wishes to upload the Course ontology. He enters the name of the file, `Course` and location of the file, `C:\Documents and Settings\SUBHA KRISHNAN\Desktop\Course.owl` on his local disk.

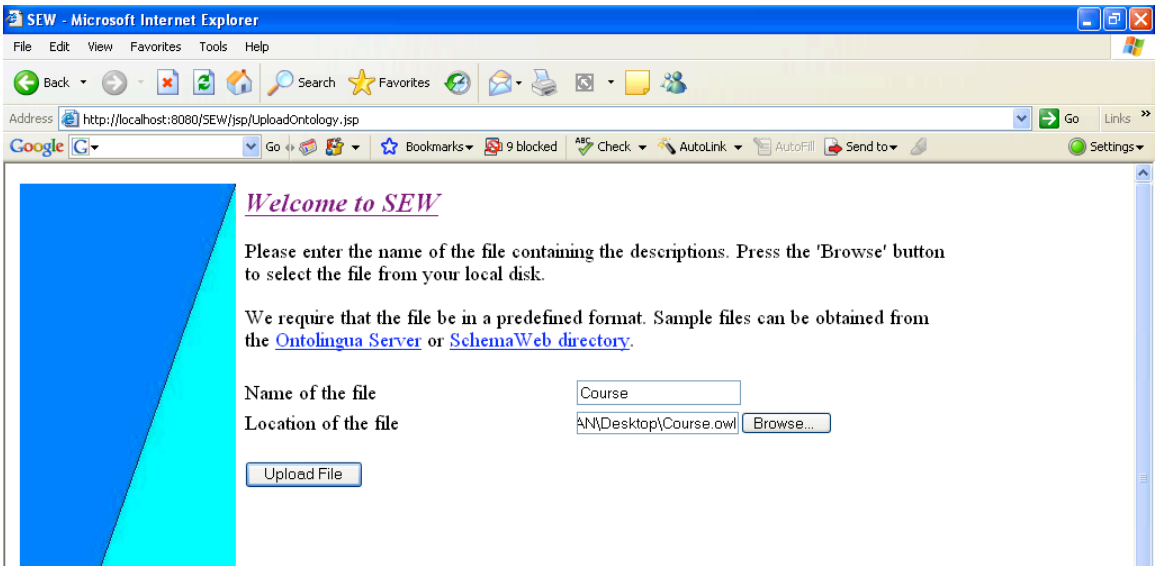


Figure 5.2 Screenshot of the user uploading the Course ontology

By clicking the `Upload File` button, the upload process is initiated.

If the file is successfully uploaded, the system displays a message to the user regarding the same.

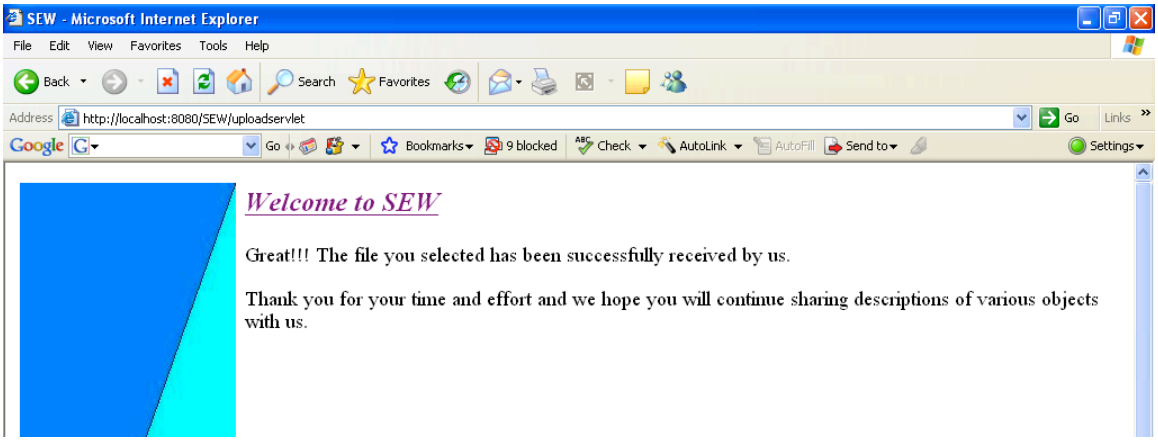


Figure 5.3 Screenshot of the status of the upload process

The Program ontology and the Association Between Program And Course ontology are also uploaded following the same procedure.

5.3 Creating the knowledge bases

The knowledge bases are created based on the ontology. The table structure in the relational database is created from the ontology and the table content, which forms the knowledge base, comes from the user input. The system asks questions to the user. The questions are generated dynamically from the ontology. User answers the questions and information extracted from the answers is stored in the knowledge base.

5.3.1 The Course knowledge base

The system displays the existing ontologies. Different users have uploaded these ontologies into the system. These ontologies describe a course, a program and the association between the two.

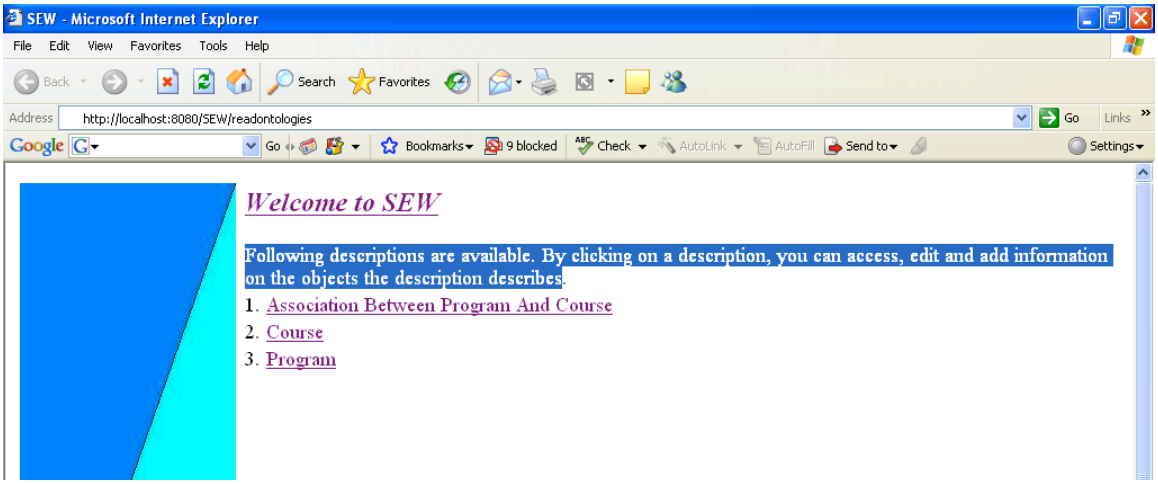


Figure 5.4 Screenshot of display of existing ontologies

By clicking on the ontology hyperlink, user gains access to the corresponding knowledge base.

For example if the user clicks on `Course` in the screen shown on the previous page and since the knowledge base for `Course` is empty, the following screen is shown:

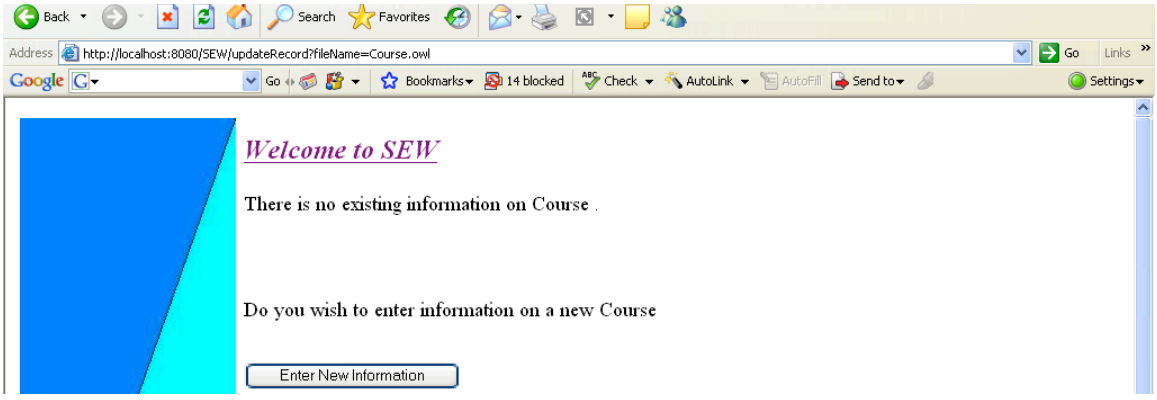


Figure 5.5 Screenshot of user selection page

Now, the user that wants to expand the knowledge base needs to select the `Enter New Information` button. Let us assume that the user selects the `Enter New Information` button.

The system extracts information from the user about the various courses by asking the following questions:

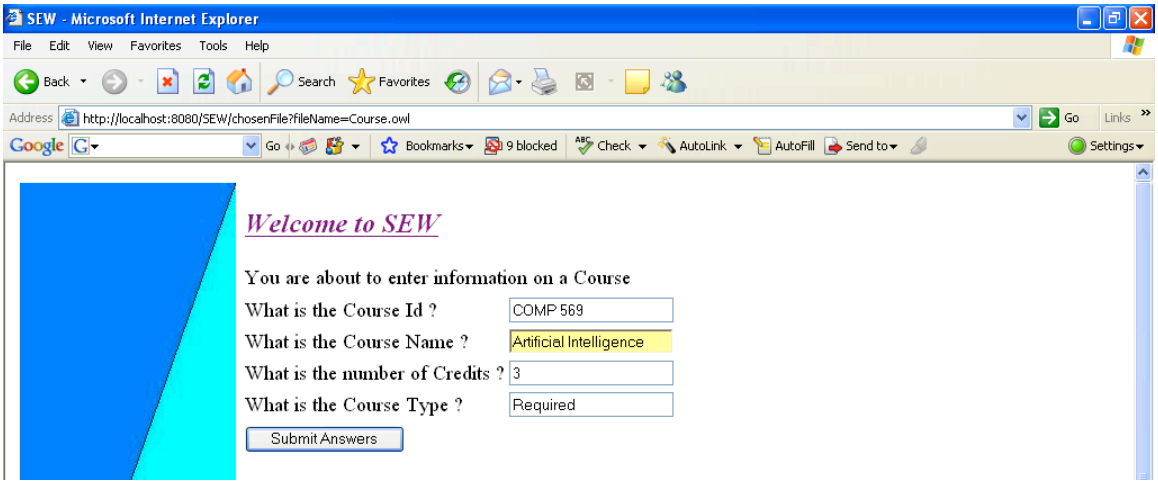


Figure 5.6 Screenshot of question-answer page for extracting information on course

User clicks on `Submit Answers` to enter his answers into the knowledge base.

We create a knowledge base for the courses offered in the CSUCI MSCS program using the question – answer module described above. The snapshot of the knowledge base thus created is as follows:

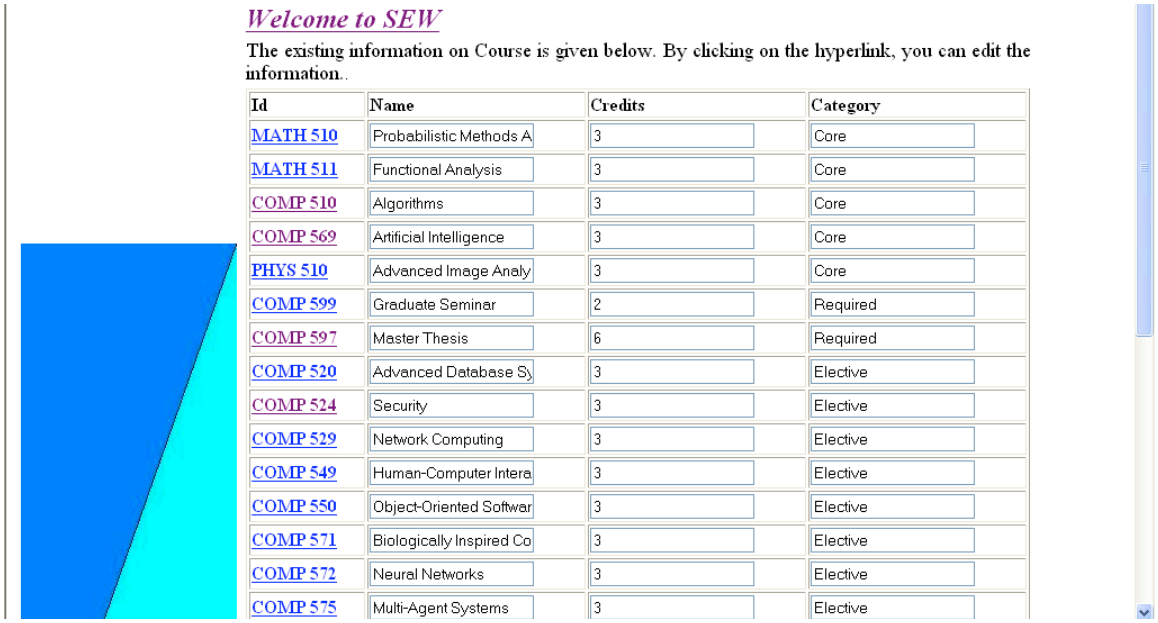


Figure 5.7 Screenshot of display of Course knowledge base

The user can click on the hyperlink to update a particular record.

5.3.2 The Program knowledge base

The knowledge base for Program ontology is created using the question – answer module given below:

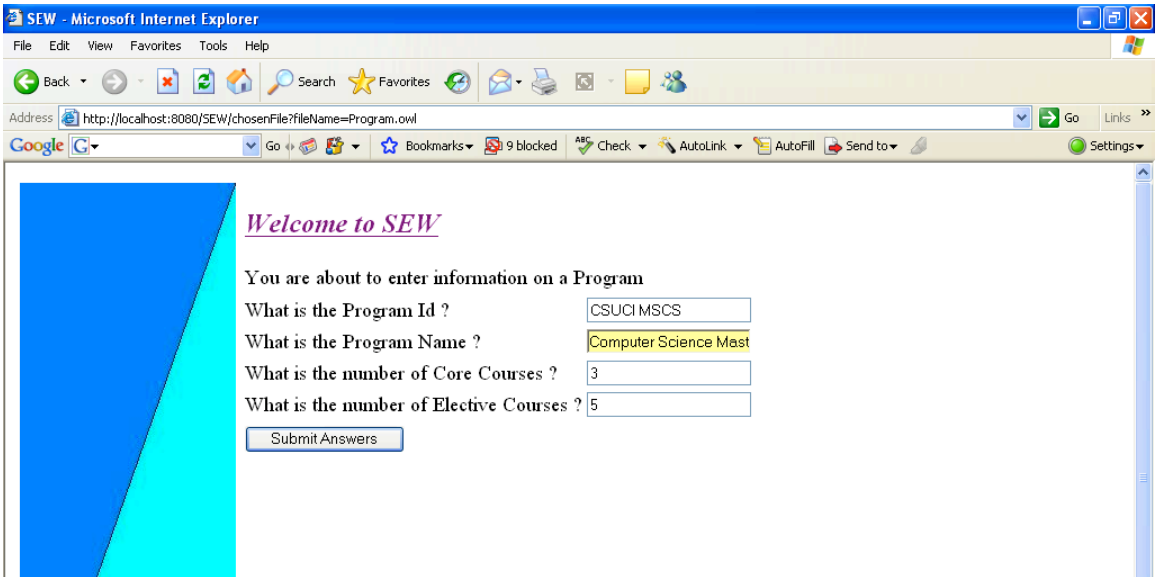


Figure 5.8 Screenshot of question-answer page for extracting information on program

5.3.3 The AssociationBetweenProgramAndCourse knowledge base

The courses are associated with CSUCI Masters program using the question-answer module given below:

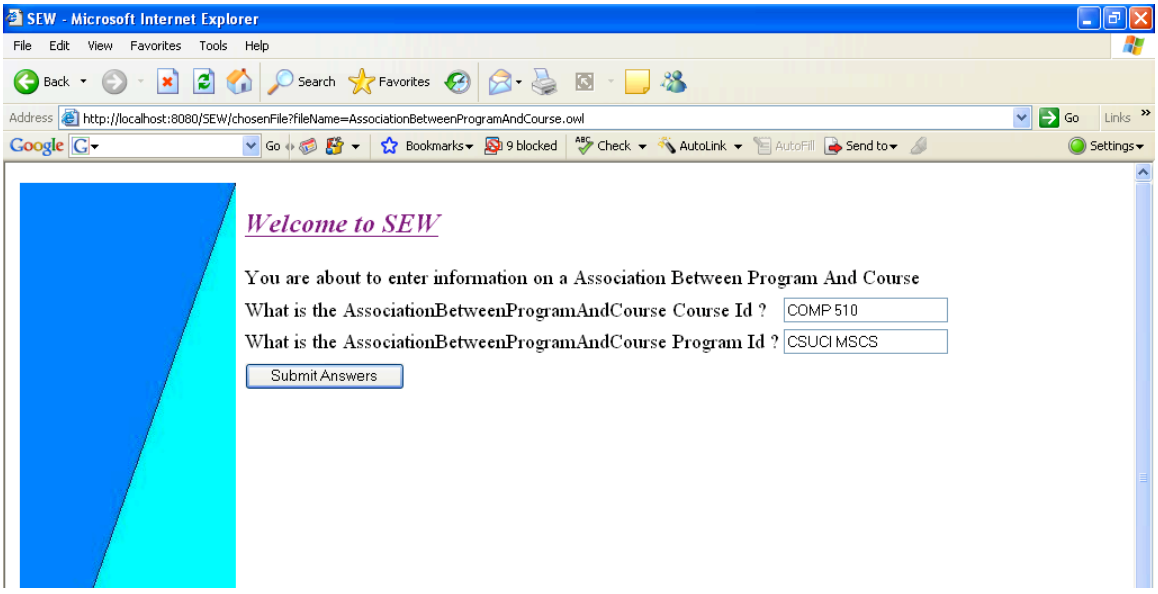
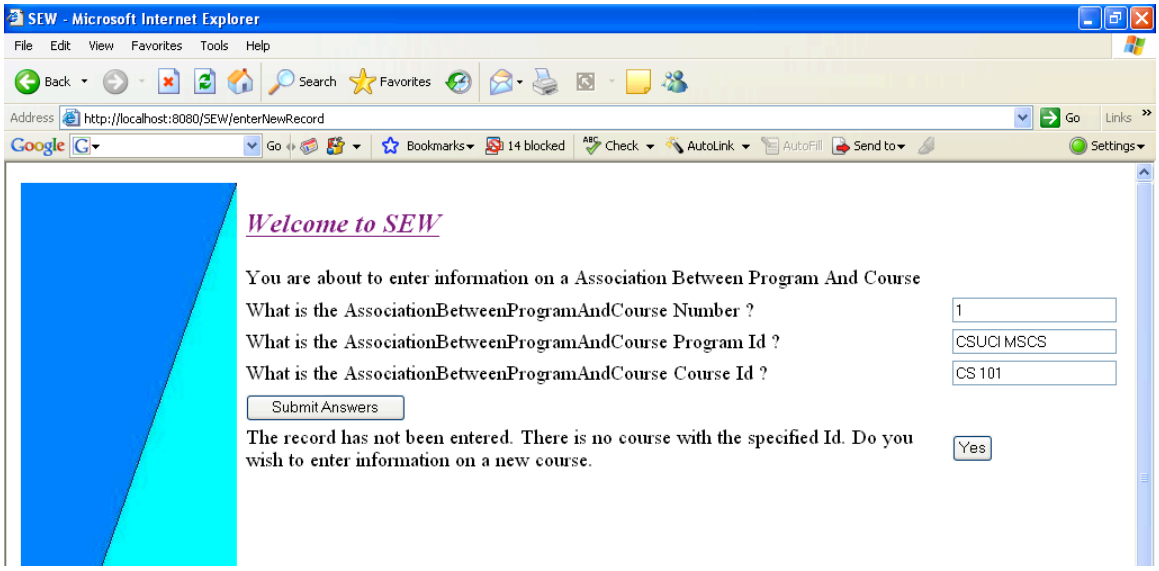


Figure 5.9 Screenshot of question-answer page for extracting information on association between program and course

If the user enters the wrong course or program, the system is able to detect it and prompt the user.



Since CS 101 does not exist, the system raises an exception. The user can either correct his answer or create a new course with id CS 101.

5.4 Creating the user application

Student is prompted to enter the program he has enrolled into and the courses he has taken. The system checks if the student has completed the required coursework and informs him whether he is eligible to graduate or not.

Student enters the id of the program from which he wishes to graduate as follows:

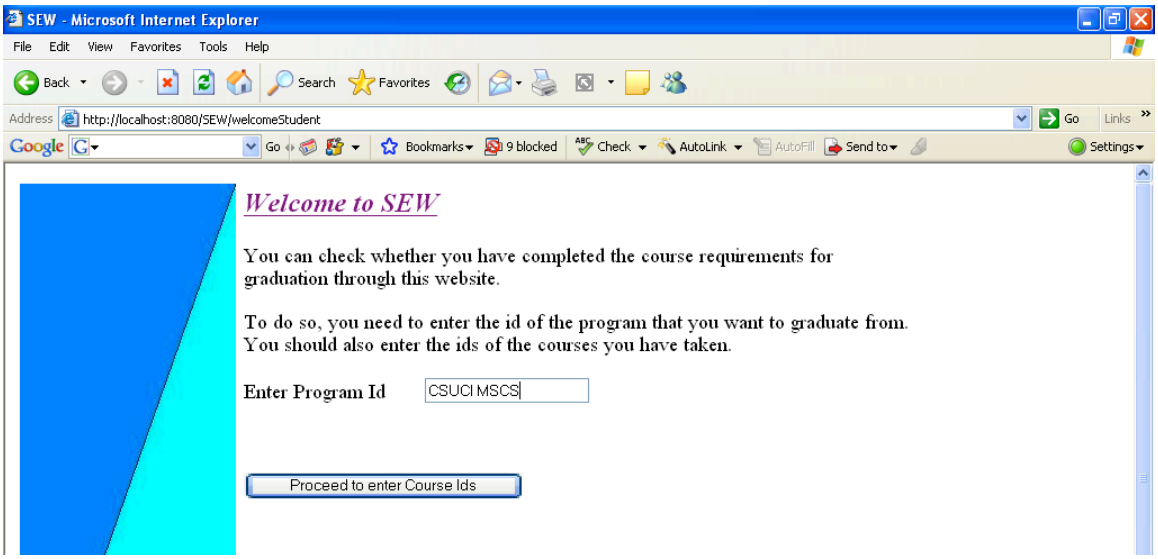


Figure 5.10 Screenshot of user inputting program id

User enters the ids of the courses he has taken:

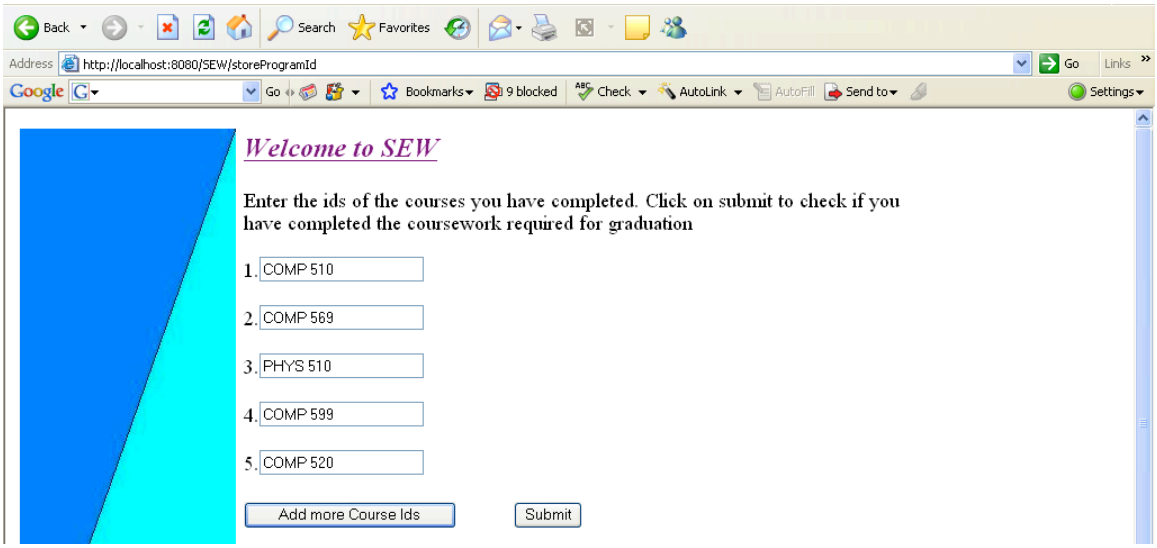


Figure 5.11 Screenshot of user inputting the courses he has taken

He clicks on `Add more Course Ids` as he wishes to add more courses as follows:

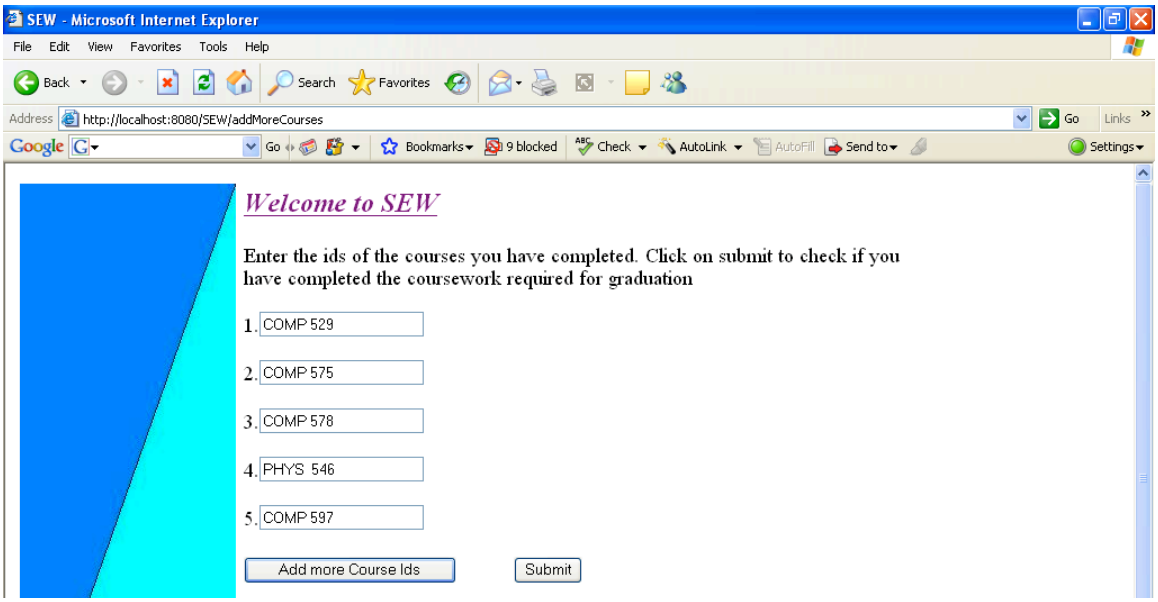


Figure 5.12 Screenshot of user inputting the courses he has taken

User clicks on `Submit` button.

The system first checks if the student has completed all the required courses for the program. Then the system checks if the student has completed the required number of core courses and elective courses. The checks are carried out against the knowledge base. In this case, the student has completed the program coursework and is eligible to graduate.

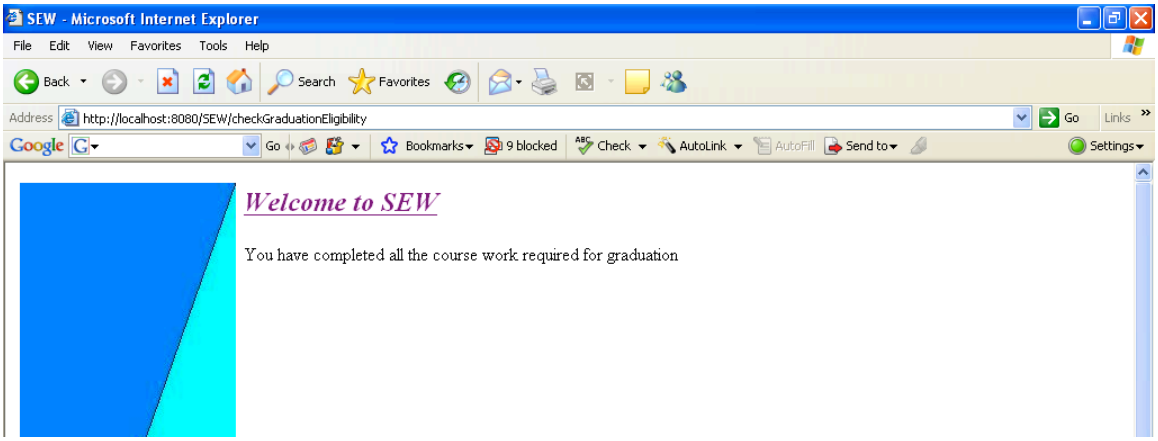


Figure 5.13 Screenshot of user eligibility for graduation

When the input is: `COMP 510, COMP 569, PHYS 510, COMP 599, COMP 520, COMP 524, COMP 529, COMP 549 and COMP 550`

The output is: `You have not completed the required course: COMP 597` (The Masters Thesis course)

When the input is: `COMP 569, PHYS 510, COMP 599, COMP 520, COMP 524, COMP 529, COMP 549, COMP 550 and COMP 597`

The output is: `You have not completed the required number of core courses`

When the input is: `COMP 510, COMP 569, PHYS 510, COMP 599, COMP 524, COMP 529, COMP 549, COMP 550 and COMP 597`

The output is: `You have not completed the required number of elective courses`

5.4 Analysis of results

SEW's ontology and knowledge acquisition process was carried out in a user friendly way. User was guided through a set of easy to use interfaces that allowed him to upload ontologies and enter knowledge. The navigation through the website was simple and the complex background processes were hidden from the user.

It was possible to express the Course, Program and Association between Course and Program ontologies in OWL and hence the ontology language, OWL was found to be sufficiently descriptive in describing the Curriculum domain.

Ontologies could be uploaded into the system by a click of a button. The details of the complex process on how the knowledge base structure is created from the ontology are hidden from the user. The information represented in the ontology was retained while creating the knowledge base structure. So the user did not lose information while transforming the classes in an OWL ontology into tables in the relational database.

The question-answer module represents a guided knowledge acquisition process. A user need not prepare a whole document on a subject whose information he want to share. SEW will guide him by asking him the questions and extracting information from his answers. As the process is simpler, he may be encouraged to share more information.

Questions were easy to interpret. The answers were editable, so the user can correct his answers or update them at later stage.

The test application, which uses SEW, checks if a student has completed the required coursework for graduation. It produced correct results for a variety of input test cases. SEW indicates not only whether the student is eligible for graduation but also which type of courses (core, elective or required) the student needs to take if he is not eligible for graduation. The test application implementation shows how the structured data in SEW along with some processing logic can enable SEW analyze and answer questions about a domain.

5.5 Limitations

1. The SEW cannot handle updated versions of the same ontology. If the SEW already has Course1.0.owl and the user uploads Course1.1.owl, the SEW does not update the knowledge base with respect to the updated version of the ontology.
2. The system does not prompt the user to key in the association between course and program in case the user does not do it.
3. Some of the user interfaces can be improved on. For example:
    a. There should be a provision to delete answers.
    b. The SEW can display the user input at the time of output while checking for graduation eligibility so that the user can edit the input information and submit it in case he made a mistake instead of entering all the data again.

Chapter 6

Future Work

In this chapter, we will discuss the issues that came up while defining the scope of the project and experimenting with it. The project enhancements listed below will make SEW more powerful in terms of collecting, retrieving and processing data.

6.1 Future Research and Implementation Work

SEW is an open-ended project. Although many improvements are possible, we will discuss those enhancements that would attract more users to contribute to the system and that would be reused in other Semantic Web projects, thus contributing to the success of the technology that attempts to change and improve the way Internet handles data.

We describe the enhancements in the following sections:

a. Migrate data from Wikipedia to SEW

Data can be migrated from Wikipedia to SEW. Text documents in Wikipedia could be linked to the appropriate ontologies and processed to increment SEW knowledge base.

b. Ability to accept updated versions of the ontology

An enhancement to the system would be to update the knowledge base when an updated version of the ontology is made available.

c. Expose data to other applications

In SEW, we are building a rich repository of information. Hence, we could build a web service as an enhancement to the system to expose the system's knowledge base to other Internet applications. Also, data belonging to different knowledge bases (different Internet applications) can be shared to make up for incomplete information.

d. Interface SEW with a reasoner

SEW can be interfaced with an OWL reasoner like the JTP reasoner [58] which can perform analysis on the knowledge base.

Chapter 7

Conclusions

The Semantic Web technology revolutionalizes the way users can access data over the Internet. SEW is a semantically enabled project that attempts to standardize knowledge representation through ontologies and knowledge bases. SEW collects ontologies by allowing users to upload them. By asking the user questions and storing his answers, SEW builds a knowledge base. Since the questions are generated from the ontology, the answers are stored along with their semantics. Hence applications can directly interpret the knowledge and analyze it to draw conclusions without human intervention.

The advantages of building such a knowledge base are demonstrated using an experiment. A university curriculum is defined using the Program and Course ontology and a relation between the two. The advantage of using ontologies within a web application is that it makes the knowledge that it represents standardized and reusable. Let us assume we have two different universities, CSUCI and CSUN using the same ontologies to define their Computer Science curriculum. The courses available in the Computer Science program of the two universities may be different. The number of courses the student needs to take may also be different. Thus the knowledge bases of CSUN and CSUCI Computer Science curriculum may be completely different. But since the structure of the knowledge base has originated from the same ontology the application that tests for graduation eligibility in CSUCI can also be used in CSUN to check if a student from CSUN has completed his university's program coursework requirements. All applications that run on any one of the university's curriculum data can run successfully on either of the university's curriculum data. Knowledge bases from one university can be migrated to another university as the knowledge representation has become standardized.

The knowledge acquisition process is a guided process where the system prompts the user with questions and extracts information from his answers. The system knows what information to expect from the user through the ontology. The user need not prepare a document or an article on a subject. He needs to answers the questions and if he answers all the questions, the information is considered complete. We thus incorporate intelligence to the system for extracting knowledge and this user friendly way may encourage the user to contribute more to the system.

The enhancements to the project include features for widening the scope of the system and making it more flexible. Semantic Extensions to Wikipedia will serve as a model website empowered by the Semantic Web that would encourage other developers to integrate this powerful technology into their own web applications.

Bibliography

1. Maurice de Kunder. “The size of the World Wide Web”. < http://www.worldwidewebsize.com/>
2. Joseph P. Bigus and Jennifer Bigus. Personal Agent Manager Application. In Margaret Eldridge, editor, Constructing Intelligent Agents Using Java, 2nd Edition, pages 247-283. John Wiley & Sons, Inc Publishers, New York, 2001.
3. David Austin. “How Google Finds Your Needle in the Web's Haystack”. < http://www.ams.org/featurecolumn/archive/pagerank.html>.
4. “Wiki”. < http://en.wikipedia.org/wiki/Wikis>.
5. Tim Berners-Lee, James Hendler and Ora Lassila. “The Semantic Web”. Scientific American Magazine. May 2001. <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21>.
6. Ivan Herman.“Semantic Web”. < http://www.w3.org/2001/sw/>.
7. Deborah L. McGuinness and Frank van Harmelen, editors. “OWL Web Ontology Language Overview”. W3C Recommendation. February 2004. <http://www.w3.org/TR/owl-features/>.
8. “Semantic Web”. < http://en.wikipedia.org/wiki/Semantic_Web>.
9. “XML Complete”. Sybex, San Francisco, 2001. ISBN: 0–7821-4033–5.
10. “RDF Tutorial”. < http://www.w3schools.com/rdf/default.asp>.
11. Michael K. Smith, Chris Welty and Deborah L. McGuinness, editors. “OWL Web Ontology LanguageGuide”. W3C Recommendation. February 2004. < http://www.w3.org/TR/owl-guide/>.
12. Dr Nicholas Gibbins and Stephen Harris, researchers. The University of Southampton, owner. “3store”. <http://www.aktors.org/technologies/3store/>.
13. Kit-ying Hui and Alun Preece, University of Aberdeen AKT Group. “An Infrastructure for Knowledge Communication in AKT Version 1.0”. <http://www.csd.abdn.ac.uk/~apreece/akt/aktbus/aktbus-1.0/index.html>.
14. University of Aberdeen, owner. “COCKATOO fact-file”. <http://www.csd.abdn.ac.uk/~apreece/akt/aktbus/aktbus-1.0/index.html>.
15. Hamish Cunningham. “GATE - a General Architecture for Text Engineering”. April 2002. < http://www.dcs.shef.ac.uk/~hamish/GateIntro.html>

16. Dr Kalina Bontcheva , Dr Hamish Cunningham, Dr Diana Maynard and Mr Valentin, researchers. The University of Sheffield, owner. “ANNIE - Open Source Information Extraction fact-file”. <http://www.aktors.org/technologies/annie/>.
17. “Annie Demo”. <http://gate.ac.uk/annie/index.jsp>
18. “George Washington”. <http://en.wikipedia.org/wiki/George_washington >.
19. Dr Fabio Ciravegna, researcher. The University of Sheffield, owner. “Amilcare fact-file”. <http://www.aktors.org/technologies/amilcare/>.
20. Dr JB Domingue and Dr E Motta, researchers. Open University, owner. “WebOnto fact-file”. <http://www.aktors.org/technologies/webonto/>.
21. Thomas Leonard, researcher. The University of Southampton, owner. “Dome fact-file”. <http://www.aktors.org/technologies/dome/>.
22. University of Aberdeen, owner. “NMARKUP fact-file”. < http://www.aktors.org/technologies/nmarkup/>.
23. Jessica Chen-Burger, researcher. University of Edinburgh, owner. “AKT Research Map fact-file”. < http://www.aktors.org/technologies/researchmap/>.
24. “The AKT Support Ontology” < http://www.aktors.org/ontology/support>.
25. “The AKTive Portal Ontology”. < http://www.aktors.org/ontology/portal>.
26. The University of Southampton, owner. “eServices fact-file”.<http://www.aktors.org/technologies/e-services/.
27. Kalfoglou, Yannis and Domingue, John and Motta, Enrico and Vargas-Vera, Maria and Buckingham Shum, Simon. “MyPlanet: an ontology-driven Web-based personalised news service”. In Proceedings IJCAI 2001 workshop on Ontologies and Information Sharing, Seattle, USA. 2001. < http://eprints.aktors.org/10/02/ontoIS01final.pdf>.
28. “Jena - A Semantic Web Framework for Java”. < http://jena.sourceforge.net/>.
29. “Pellet”. < http://www.mindswap.org/2003/pellet/>.
30. “Pellet Online Demo”. < http://www.mindswap.org/2003/pellet/demo.shtml>.
31. “IBM Integrated Ontology Development Toolkit**”.** July, 2004. < http://www.alphaworks.ibm.com/tech/semanticstk>.
32. “KAON2”. <http://kaon2.semanticweb.org/>.
33. “Protégé”. <http://protege.stanford.edu/>
34. Domingue, J., Motta, E. and Corcho Garcia, O. “Knowledge Modelling in WebOnto and OCML: A User Guide”. 1999. <http://kmi.open.ac.uk/projects/ocml/ocml-webonto-guide.zip>.
35. Peter Troxler, Andy Aiken and Derek Sleeman. University of Aberdeen, owner. “Refiner ++ User Manual”. September 2004. < http://www.csd.abdn.ac.uk/~aaiken/refiner/manual.pdf >.

36. Derek Sleeman, researcher. University of Aberdeen, owner. “ReTAX+ fact-file”. < http://www.aktors.org/technologies/retax/>

37. Christopher Brewster, researcher. The University of Sheffield, owner. “Adaptiva fact-file”. <http://www.aktors.org/technologies/adaptiva/>

38. Christopher Brewster, Fabio Ciravegna and Yorick Wilks. “User-Centred Ontology Learning for Knowledge Management”. <http://eprints.aktors.org/125/01/brewster_nldb02.pdf>.

39. “The Ontolingua Server” <http://www.ksl.stanford.edu/software/ontolingua/>

40. “The SchemaWeb directory”. <http://www.schemaweb.info/default.aspx>.

41. Dr JB Domingue, Mattia Lanzoni, Dr E Motta and Maria Vargas-Vera, researchers. Open University, owner. “Semantic Annotation with MnM fact-file”. <http://www.aktors.org/technologies/mnm/>.

42. “Wikipedia:About”.<http://en.wikipedia.org/wiki/Wikipedia:About_Making_the_best_use_of_Wikipedia>

43. “Eclipse”. <http://www.eclipse.org/>.

44. Scott Storkel. “An Introduction to the Eclipse IDE”. December 2002. <http://www.onjava.com/pub/a/onjava/2002/12/11/eclipse.html>.

45. “MySQL”. <http://www.mysql.com/>.

46. “Tomcat”. <http://tomcat.apache.org/>.

47. Tony Sintes, JavaWorld.com. “App Server, Web Server: What’s the difference?”. August 2002. <http://www.javaworld.com/javaqa/2002-08/01-qa-0823-appvswebserver.html?page=1>.

48. “Configuring & Using Apache Tomcat”. <http://www.coreservlets.com/Apache-Tomcat-Tutorial/#Development-Environment>.

49. “Java”. http://java.sun.com/.

50. “RDFReactor”. <“http://rdfreactor.ontoware.org/”>

51. Max Völkel, FZI Forschungszentrum Informatik, Karlsruhe, Germany. “RDFReactor – From Ontologies to Programmatic Data Access”. <http://www.xam.de/2006/05-RDFReactor-JUC2006.pdf>

52. “Velocity”. <http://velocity.apache.org/>.

53. “Java Tutorial on Reflection”. 1995-2006 Sun Microsystems, Inc < http://java.sun.com/docs/books/tutorial/reflect/>.

54. “Java Tutorial on JDBC Database Access”. 1995-2006 Sun Microsystems, Inc < http://java.sun.com/docs/books/tutorial/jdbc/>.

55. Marty Hall. “A tutorial on Java Servlets and JSPs”. 1999. <http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/>

56. Ramesh Kumar Swarnkar. “Java/JSP upload”. August 2004. <http://forums.codecharge.com/posts.php?post_id=44078>.

57. “GATE”. <http://gate.ac.uk/>.

58. “JTP reasoner”. <http://www-ksl.stanford.edu/projects/wine/explanation.html>.

59. “CSUCI Master of Science in Computer Science, Extended Education”. <http://www.csuci.edu/exed/mscompsci.htm>