A System for Management of Semantic Data (Ontology Components) in Semantic Web

M O H A M M A D R E Z A R A J A E I

Master of Science Thesis
Stockholm, Sweden 2007

ICT/ECS-2007-23

A System for Management of Semantic Data (Ontology Components) in Semantic Web

By

M O H A M M A D R E Z A R A J A E I

rajaei@kth.se

Royal Institute of Technology
Stockholm, Sweden

S u p e r v i s o r a n d E x a m i n e r

V L A D I M I R V L A S S O V

vladv@kth.se

Associate Professor, PhD
(ECS/ICT/KTH)

Master of Science Thesis
Stockholm, Sweden 2007

ICT/ECS-2007-23

Abstract

Today, the information on the Web is designed for human interpretation and it is not machine processable. Thus according to the inventor of the Web, Tim Berners-Lee, the Web, has not achieved one of its primitive goals: being useful for the machines.

In order to achieve this goal, Semantic Web is introduced as a vision for the future of the Web. Its approach is to develop languages and methods to express information on the Web in processable, understandable, and useable forms for machines, as well as human beings. In this approach, some standards and languages are defined by W3C such as Resource Description Framework (RDF) as a data model framework, RDF Schema as a vocabulary description language, and Web Ontology Language (OWL) as a way to represent the explicit meaning and relations of the terms used in vocabularies.

However, Semantic Web is still in its early steps and necessary tools are required to facilitate dealing with Semantic Data and building Semantic enabled applications. The main goal of this thesis is to develop a Web based Ontology management system based on Jena Semantic Web framework. The developed application, UltimateOMS, enables the users to create, manipulate, and manage Semantic data and Ontology components in forms of RDF, RDF Schema, and OWL through a web based user interface. UltimateOMS will bring the necessary features such as visualization graph of Semantic data, storing Semantic data in different databases, together with many other features to facilitate the process of managing Semantic data for users who deal with Semantic data.

Keywords: Semantic Web, Semantic data, Ontology, RDF, RDF Schema, OWL, UltimateOMS, Jena, IsaViz, Graphviz, JSP, Java Servlet.

Acknowledgements

I would like to express my acknowledgment to my supervisor and examiner, Associate Professor Vladimir Vlassov for his excellent support and guidelines during this project.

I also would like to express my acknowledgment to Fredrick Lekarp for his support during this project and my studies in Stockholm.

Most of all, I would like to thank my wife Mina for her constant support and encouragement.

Table of Content

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

> *"The Web was designed as an information space, with the goal that it should be useful not only for human-human communication, but also that machines would be able to participate and help." [1]*
>
> *Tim Berners-Lee, the inventor of World Wide Web*

Today, most information on the Web is not machine processable and therefore, according to the inventor of the Web, Tim Berners-Lee, this Web which is widespread around the world now has not achieved one of its primitive goals which was being useful for the machines.

A major obstacle to reach this goal was the fact that machines or computers have different needs, comparing to human beings, in order for them to "understand" the information on the Web. In fact, most available information on the Web is mainly designed for human interpretation, which makes it unusable for machines. Even the information, which is derived from a structured database with meaningful columns, is not well defined enough for a machine to be able to understand and use it. Therefore, in order to achieve the Web's primitive goal, information on the Web needs to be expressed in a form

that machines would be able to understand it instead of simply displaying it. [1]

In this concept, when the term "understand" for machines is used, it does not imply to some artificial intelligence empowering machines with magical abilities. In the contrary, it relies on today's ordinary machines' capabilities, which can perform well-defined operations based on well-defined data to solve well-defined problems. [2]

In fact, Semantic Web approach does not need revolutionary machines, which understand the human beings; it needs human beings to make some extra effort to write the Web's language in a better-defined way that would make it possible for the machines to use the Web. [2]

Consequently, the Semantic Web approach is to develop languages and methods to express information on the Web, which is processable, understandable, and useable for machines as well as human beings.

The Semantic Web is a vision for the future of the Web. In this vision, information is given explicit meaning, which makes it possible for machines to perform integration, processing, and understanding of information on the Web. [3]

An important component of Semantic Web, which is a way of representing semantics and enabling them to be used by machines and specifically by web applications, is called ontology. Ontology is a way of giving explicit meaning to information by structuring and defining the meaning of metadata. For each specific subject or area of knowledge, ontology defines the terms, which are used to represent and describe that subject. Moreover, ontology defines the relationships among the basic concepts in areas of knowledge and also defines computer usable definitions of those basic concepts. [4]

Using ontology, web applications would be able to understand the semantics of documents and therefore become capable of processing,

performing integration, and understanding them. This will make the Web useful and understandable for web applications and therefore machines.

1.2 Motivation of the Project

Since Semantic Web is in its early steps, there is long way to make all necessary standards of the Semantic Web. The W3C is currently responsible for publishing Semantic Web standards and different wok groups are currently working as part of the Semantic Web activity. Meanwhile, different types of tools are provided based on published Semantic Web standards by different vendors in order to fulfill users' needs for creating and managing Semantic data.

Nevertheless, to the best of our knowledge, none of the existing solutions provides a complete tool containing all required features for users dealing with semantic data, which will result in a slow and timely cost process.

The motivation of this project is to facilitate creating and management of Semantic data by developing a new Web based application containing all necessary features. The new system will bring all necessary functions for managing semantic data in one place thus making it much easier for users to create and manage their Semantic data.

1.3 Project Goals and scope

The goal of this project is to develop a Web based application based on one of the existing frameworks for building Semantic Web applications. The application will enable all users to create and manage their Semantic data (Ontology components) in RDF, RDF Schema, and OWL formats through a web based user interface and unlike the

existing tools; it provides the necessary features in one place to facilitate process of managing Semantic data.

The developed application will provide different features such as creating, uploading, and importing Semantic data including RDF (Resource Description Framework) and OWL (Web Ontology Language), storing Semantic data in different types of databases, designing and editing capabilities of Semantic data (include creating, editing, and deleting), inferring, querying, visualizing graph of Semantic data.

Comparing the existing frameworks and the issues related to the framework including the performance is out of the scope of this project.

1.4 Related Work

pOWL [5] and Sesame [6] are two web-based tools for managing Semantic data and Protege [7] is one of the many client based ontology editors and it is not web based.

pOWL is a web-based application for editing and managing knowledge for Semantic Web, which is developed based on PHP and MySQL. It supports browsing, editing, and querying of RDF Schema and OWL ontologies but there is no facilities for storing Semantic data in different databases, reasoning, and visualization graph of Semantic data.

Sesame is an open source framework for RDF and RDF Schema with inferring and querying capabilities. Different types of storage system such as relational databases, file systems, and in-memory can be used along with Sesame. The web-based tool of Sesame includes browsing and querying features without editing and managing capabilities of Semantic data and visualization graph.

Protege is a well-known, sophisticated, and open source ontology editor with support of RDF, RDF Schema, and OWL, which enables users to create and manage Semantic data. Protege is a client-based tool and its visualization capability is based on a simplified hierarchy structure and does not visualize in form of graphs, nodes, and arches.

1.5 Structure of the Thesis

This thesis consists of seven chapters. First chapter briefly gives some background information about Semantic Web, presents motivation of the project, project goals, and finally introduces some related works. Second chapter provides information regarding concepts of Semantic Web including benefits of Semantic Web like knowledge integration, knowledge searching and inferring. This chapter also describes RDF, RDF Schema, OWL, and RDF query languages. Third chapter presents vision document of the project and defines requirements and high-level features of the developed tool. Chapter four reviews the technologies and concepts related to the architecture of the software including JSP technology and Java Servlets, Jena Semantic Web Framework, graph visualization, and finally presents the architecture of the proposed solution for implementing the Ultimate Ontology Management System (UltimateOMS). Chapter Five includes some detail information about the implementation of UltimateOMS including development platform, system configuration, session management, graph visualization, and user interface. Chapter six is about validation of the implemented use cases. As a final point, chapter seven presents conclusion of the project and gives some suggestion for future work.

Chapter 2

Semantic Web

2.1 Overview

Semantic Web is a project in the World Wide Web Consortium (W3C) under the direction of the inventor of the Web, Tim Berners-Lee, where a dedicated team works to improve, extend, and standardize the system. According to him, evolving into Semantic Web is the way that the Web can reach its full potential.

Semantic Web extends the capabilities of the Web by adding computer processable meaning to it through the use of standards, mark-up languages and related processing tools. Tim Berners-Lee defines Semantic Web as follows: "Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation". [9]

In Semantic Web, the idea is having the data defined and linked to each other as a globally linked database and provide a universally accessible platform that not only people, but also automated tools can share and process data. This way the data can be effectively used for automation, integration and also in order to reuse in different applications. [8] [9]

Although many languages, publications, and tools are already developed and published, Semantic Web technology is still in its early stage and despite the fact that the future of Semantic Web appears to be bright, yet there is no general agreement about a promising direction or characteristics of the early Semantic Web. [8] [10]

2.2 Rationale

Lack of a global system for publishing data in a way processable for any system, makes it difficult to use the Web in large scale. Data is mostly hidden in HTML documents which are only useful to some extend and in some specific contexts and not useful for other contexts. [10]

Semantic Web makes it possible to publish data in a broadly processable form. According to W3C, this will make more people willing to publish data in this format and shortly the number of Semantic Web applications will increase dramatically. These Semantic Web applications would be useable for variety of tasks and helps to increase the modularity of applications on the Web. [10]

2.3 Benefits

Currently, the Web is designed to be used by people and not by computers and machines. Semantic Web makes the data in web pages understandable for computers as well as human beings so the computers would be able to search websites and perform actions in a standardized way. This enables computers to utilize the enormous amount of services and information on the web. [8]

Moreover, a lot of services that are already possible to implement in current web will be much easier to implement with the standards introduced in Semantic Web. [8]

The benefits of Semantic Web can be categorized as knowledge integration, knowledge creation and storage, knowledge searching, knowledge inference and knowledge perspective. Below is the description of each category:

2.3.1 Knowledge integration

One of the important benefits of Semantic Web is information and knowledge integration that it brings. In order to have knowledge integration, the integration mechanism should have a distributed nature itself; otherwise, the integration would not be possible. For example, it is not realistic to hope to build a single database or XML file that integrates all the information in the Internet. Only a distributed way of integration is appropriate for the distributed nature of the Internet.

Location: A good information integration should have a mechanism to let the user know where the data resides and should be able to reach it. Semantic Web has addressed this issue by using the Uniform Resource Identifier (URI). By labeling all the sources with URIs, Semantic Web leverages from the benefits of this good old format and therefore all Semantic Web sources are findable in their unique locations.

- **Protocol**: In order to interact with data, a protocol should be in place to serve as an exchange language. Semantic Web uses standard web protocols such as HTTP, which is an easy flexible exchange language based on request and response.

- **Format**: Semantic Web uses the OWL Web Ontology Language, which is a standard data format based on Resource Description Framework (RDF) and XML. The format is very comprehensive and fulfilled the requirements of a distributed integration, which are being comprehensive and translatable.

- **Reliable**: Information and knowledge integration needs to have a mechanism to make sure the records are timely and reliable. Since Semantic Web deals directly with the actual source of data, there is no need for complex synchronization unless the due to the performance or other requirements actual source of data would not be used.

- **Purpose:** The challenge in knowledge integration is aligning the data together with the purpose. It is easy to combine data without the meaning attached to it. The beauty of Semantic Web is in bringing the meaning together with the data and this makes Semantic Web perfect for information and knowledge integration. It makes it possible to treat the data for the real meaning that it represents not just as meaningless bits. [11]

2.3.2 Knowledge creation and storage

Semantic Web encapsulates knowledge so it becomes easy to share knowledge and also change or develop the knowledge no matter it is your knowledge or someone else's shared knowledge. [11]

Traditional databases, store and share "data" and the real knowledge or meaning of that data is either in the application or in the mind of user. The knowledge, which resides in the application is very specific to that application's use. Sharing application components is an attempt to share the knowledge residing in the application. However since the application's knowledge is very narrow and specific to its own purpose, this attempt fails in reality.

In Semantic Web, knowledge can grow in many ways.

- **Horizontally**: By adding new attributes or adding and peer relationships, knowledge grows horizontally.

  Adding "date of birth" to "person" which is an attribute of that person or adding "manager" relationship between two "employees" which is a peer relationship between them are examples of horizontal growth of knowledge

- **Vertically:** Growing knowledge through inheritance is a vertical growth.

For example, an “employee” is a type of “person”. “Person” has some attributes like “name”, “place of birth”, and “employee” has those attributes as well because it is a type of person. By adding a new attribute to “person”, like “date of birth”, the same attribute becomes added to the “employee” by inheritance. This is a horizontal growth for “person” and a vertical growth for “employee”.

- **Constraints**: Knowledge can grow by adding some constraints in order to define the context.

  For example a “person” can be defined as “tall” where “tall” is a person taller than 170 cm.

Knowledge can be created, developed and can grow horizontally, vertically or by constraints remotely from anywhere in the network by referencing the knowledge using its URI. [11]

2.3.3 Knowledge searching

The goal of any knowledgebase is getting useful results for a search operation whether it is a simple question or a complicated query. Currently, there are two main ways for searching: database queries and keyword matching.

Database queries are very powerful for searching in a well-known structured environment of an individual database but useless outside that specific structure of the database.

Keyword matching in the other hand is not coupled to any structure but it can result in too many hits for a search and it is very weak in answering specific questions. For example, a keyword search is useless in answering: “who was the manager of the research department in March 2006”.

Semantic Web brings a compromise between those two methods. It uses enough structure to support answering specific questions and also it has enough flexibility so that it is not too much couples to the underlying structure. Consequently, one does not need to know the structure beneath, to be able to get good result for a search.

Queries are independent from the knowledge structure. Therefore, they can stay the same even if the underlying knowledge structure keeps changing. [11]

2.3.4 Knowledge inference

Semantic Web has the ability to fill in the missing pieces by deducting from the related data.

For example, given that a person "Mary" refers to another person "Mark" as her father, the system will infer that "Mark" has a daughter "Mary". This technique can bring very useful conclusions because it deals with the meanings and semantics rather than data and bits. This is a very unique characteristic and gives a big advantage over the traditional systems. [11]

2.3.5 Knowledge perspectives

Semantic Web makes it possible to have knowledge aligned with one's specific needs and domain of interest. Traditional systems, force the user to align himself/herself to the single point of view represented in that specific system. However, Semantic Web enables selective integration of knowledge and construction of new knowledge and as a result creating new knowledgebase according to any need. This is possible by building upon the existing knowledgebase or starting one from the scratch.

Knowledge perspective is the ultimate goal of Semantic Web. [11]

2.4 Semantic Web Stack

Semantic Web technologies are arranged into layers shown in Figure 2.1. The two base layers are technologies used in the current Web. The next six layers build Semantic Web on those two inherited layers and the top one adds trust and completes a Semantic Web of trust. The complexity increases from the bottom to the top. The functionality of higher layers depends on lower layers.

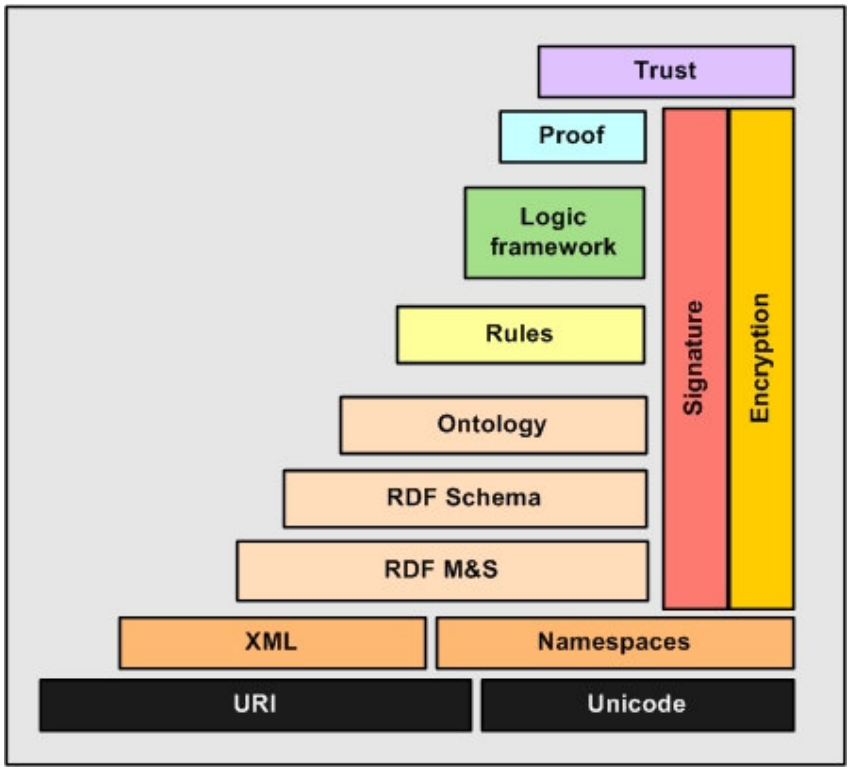

Figure 2.1: Semantic Web Stack, from Tim Berners-Lee presentation for Japan Prize, 2002

- **Layer 1: URI and UNICODE**

  URIs are global identifiers that uniquely identify information. Current Web uses URL while Semantic Web uses URIs. Unlike URLs, URIs do not necessarily retrieve any information. Unicode

is a global character-encoding standard, which supports international characters.

Those two technologies, Unicode and URI, which are taken from the current web standards, provide the global characteristic, for Semantic Web.

- **Layer 2: XML and Namespaces**

Semantic Web should easily integrate with the current Web. HTML does not have the ability to include everything needed for Semantic Web. XML has more abilities and is superset of HTML.

Namespaces increase the modularity of XML and also increases the ability to reuse XML vocabularies together with XML schemas. In Semantic Web, Namespaces are used for the same purpose.

- **Layer 3: RDF Model and Syntax**

This is the first layer developed for Semantic Web specifically. Semantic Web is built on RDF which itself is built on syntaxes that use URIs to represent data. Semantic Web is about representing data and not presenting data. This data representation is usually in triple based structures that can be stored in database or interchanged on the Web using a set of syntaxes. These syntaxes, which are developed for this task, are called Resource Description Framework (RDF). [10]

This layer is detailed in the subsection 2.5.

- **Layer 4: RDF Schema**

  Despite the fact that RDF provides the tool to build semantic networks, it does not provide all semantic network facilities needed for Semantic Web. RDF Schema provides some facilities to define metadata vocabularies similar to Object Oriented constructs and also to implement them in a modular way similar to XML schemas.

  RDF Schema is detailed in section 2.6.

- **Layer 5: Ontology**

  Generally, the representation of the terms and identifying their relationships is called Ontology. OWL is the language developed by World Wide Web Consortium for web ontologies, which makes it possible to represent the meaning of terms that are used in vocabularies and also relationships between those terms. Metadata vocabularies defined in RDF Schemas can be considered simplified ontologies.

  OWL and Ontology is detailed in section 2.7.

- **Layer 6: Rules**

  In this layer, dynamic knowledge is captured as a set of conditions that must be fulfilled to be able to achieve the result or the set of results in the rule.

  The technology behind this layer is Semantic Web Rule Language (SWRL), which is based on Rule Modeling Language or RuleML. Similar to RuleML, SWRL is a very comprehensive language and covers all sorts of rules including derivation, transformation, and

reaction rules. SWRL can specify queries and inferences in Web ontologies and covers mappings between Web ontologies as well as dynamic Web behaviors of workflows, agents, and services.

- **Layer 7: Logic**

Semantic Web needs to have a powerful logical language for inference. The purpose of this layer is providing the features of First Order Logic (FOL).

There are some alternatives and different languages have been considered. One of the first alternatives was RDFLogic, which provides some extension to basic RDF. Another more recent one is SWRL FOL, which is an extension of the rule language SWRL to cover FOL features.

For Semantic Web to become expressive enough to help us in a wide range of situations, it will become necessary to construct a powerful logical language for making inferences.

- **Layer 8: Proof**

The idea in this layer is to write down the proofs for the problem. Using inference engine rather than the traditional way of black-box principle used in computer programs, makes Semantic Web open and therefore the inference engine can be asked to provide proofs for the conclusion.

- **Layer 9: Trust**

This layer uses all the layers below together with encryption and signature in order to build the Semantic Web of trust. Encryption and signature are technologies already available in current Web

and the trust layer makes use of them to be able to trustfully bind statements with their responsible parts.

Therefore, the Semantic Web of trust will use Public Key infrastructure, which is already in place and Semantic Web of trust, might be able to contribute to this infrastructure by using a more distributed structure and removing the rigidity, which is a consequence of its hierarchical structure.

As a result of using the encryption and signature together with this layer, trust engines can be constructed, which include reasoning engines together with digital signatures. Using these trust engines, the Semantic Web of trust can be developed where rules can be trusted depending on the signer. [12]

2.5 RDF (Resource Description Framework)

This section presents a brief overview about the concepts of RDF. [16]

2.5.1 Overview

Resource Description Framework (RDF) is W3C proposed framework for representing information, particularly metadata about Web resources about resources, in the World Wide Web [17]. Using RDF information can be exchanged easily between applications without loss of meaning. This becomes important for those information that need to be processed and not just displayed in a simple format like HTML.

RDF uses URIs for identifying things in the Web and describes resources with properties and property values pairs. As a result, statements about resources can be represented in RDF as a graph containing nodes and arcs in which arcs represent properties of resources and nodes represent resources and property values [18].

The statements are expressed as simple triples like *<S,P,O>* in RDF. The triple *<S,P,O>*, means that subject *S*, which is a resource indicated by URIs has property *P*, which is also a resource indicated by URIs with value *O* and *O* is either a URI or literal value. Some basic properties such as type and class are defined in RDF, RDFS, and OWL [19].

2.5.2 Basic Concepts

As mentioned in section 2.2.1, the basic structure of any expression in RDF is a triple statement consisting of a Subject, Predicate, and Object. Each triple can be modeled as a directed graph using a node and an arc diagram. A collection of triples is called RDF graph.

As shown in figure 2.2, each triple is modeled as a node-arc-node link in which the *Subject* and *Object* are nodes and *Predicate* (or *Property*) is a link that always points toward *Object* and describes the relationship between *Subject* and *Object*.

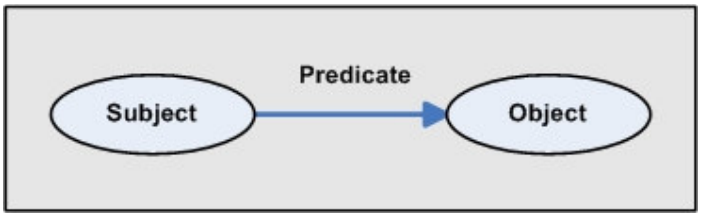

Figure 2.2: RDF triple modeled as a directed graph

In RDF graph, a *Subject* node can be a URI reference or a blank node and an *Object* node can be a URI reference, a blank node, a plain literal, or a typed literal. *Predicate* or *Property* is only a URI reference [17].

- A “URI” is a more general form of the URL (Uniform Resource Locator) and can be used to identify anything that needs to be referred. For example, below line is a URI:

  `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`

- A “blank node” is a unique node without intrinsic name that can be used in RDF statements.

- A “plain literal” is a string with an optional language tag, which is used to identify values such as dates and numbers in lexical format. For example the below line is a plain literal and indicates that the string is expressed in English (en).

  `“This is a plain literal”@en`

- A “typed literal” is a string with a URI datatype, which is used to identify values by means of lexical format. For instance, the below line is a typed literal indicates that datatype of the value “20” is integer:

  `“20”^^http://www.w3.org/2001/XMLSchema#int`

As an example, consider the following RDF code:

```
<rdf:RDF
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:dc="http://purl.org/dc/elements/1.1/"
       xmlns:foaf="http://xmlns.com/foaf/0.1/" >

<rdf:Description rdf:about="http://www.bogus.com/index.html">
  <dc:creator rdf:resource="http://www.bogus.com/staffid/10"/>
</rdf:Description>

<rdf:Description rdf:about="http://www.bogus.com/staffid/10">
  <foaf:name>Bob</foaf:name>
</rdf:Description>
```

```
</rdf:RDF>
```

The above RDF statements indicate that subject *http://www.bogus-.com/index.html* is created by object *http://www.bogus.com/staffid/10* whose name is *Bob*. There are two different triples in the above statements and each one has its own subject, predicate, and object. In the first triple, the subject is *http://www.bogus.com/index.html*, the predicate is *http://purl.org/dc/elements/1.1/creator*, and the object is *http://www.bogus.com/staffid/10*. In the second triple the subject is *http://www.bogus.com/staffid/10*, the predicate is *http://xmlns.com/foaf-/0.1/name*, and the object is *Bob*.

The RDF statements discussed above are shown as RDF graph in figure 2.3.

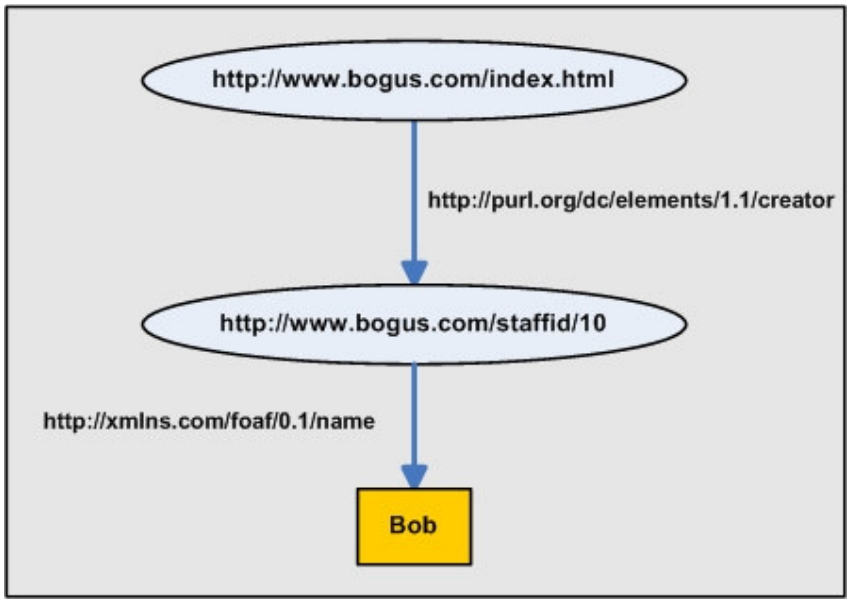

Figure 2.3: RDF graph of example

Sometimes it is necessary to describe other RDF statements using RDF. For instance, some RDF applications need to keep information about

who made RDF statements, when statements were made. RDF introduces "Reification", which is a way to provide description of the statements using RDF built-in vocabulary consisting of the type *rdf:Statement*, and properties *rdf:subject*, *rdf:predicate*, and *rdf:object* [18].

As a result, by using reification, RDF statements can be used as a resource in other statements allowing nested statements in RDF graph [20]. Figure 2.4 shows an example of a reification statement used in order to make a new statement. In this example, staff member with identifier *20*, http://www.bogus.com/staffid/20, claims that the creator of the HTML page *http://www.bogus.com/index.html* is staff member with identifier *10*, *http://www.bogus.com/staffid/20* whose name is *Bob*.

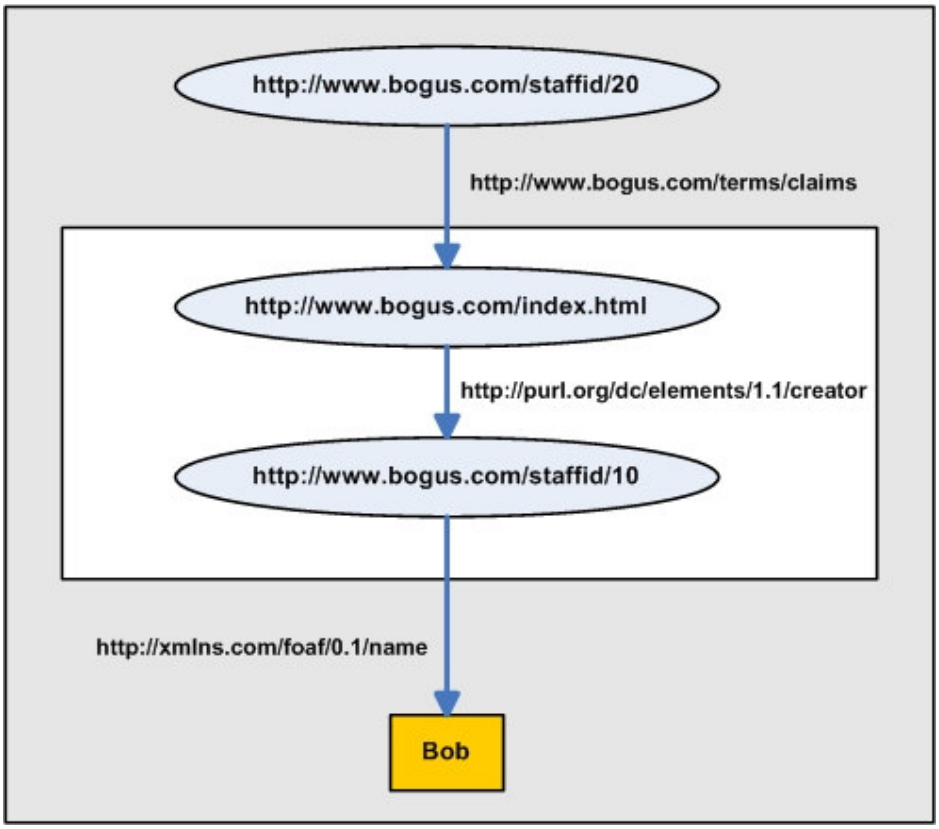

Figure 2.4: Making a new RDF statement using reification

2.5.3 Serialization Formats

RDF graphs can be encoded in different formats. The W3C has defined an XML based syntax for RDF graphs called RDF/XML, which is the standard interchange format on Semantic Web. An example of this format is mentioned in the example of section 2.2.2, which is expressed in RDF/XML. A complete description of the RDF/XML syntax specification is available on the W3C web site [21].

RDF/XML is not the only format for encoding RDF graphs, there are some other plain text formats such as N-Triples [22], Turtle (Terse RDF Triple Language) [23], and Notation3 [24].

2.6 RDF Schema (Resource Description Framework Schema)

2.6.1 Overview

RDF is a standard for building data models and it is necessary to have another layer for building specific vocabulary for those data models. RDF Schema is introduced by W3C as RDF's vocabulary description language. RDF Schema vocabulary descriptions are written in RDF and allow designer to define the vocabulary, which is used by RDF data model [25].

RDF Schema contains some predefined semantic terminology such as *Class* and *subClassOf* where using *subClassOf* property allows expressing the hierarchy of classes. As an example, if "Person" is defined as a *class*, "Staffs of a bogus company" is defined as a *subClassOf* of the "Person", and "Bob" is defined a "Staffs of a bogus company", then due to the semantics of the RDF Schema, it is implicitly true that "Bob" is also type of the "Person".

The core vocabulary in RDF Schema is defined in a namespace called *rdfs*, which is identified by the URI *http://www.w3.org/2000/01/rdf-schema#.*

2.6.2 Modeling Primitives

This section presents the main classes, properties, and constraints of RDF Schema. A complete description of these primitives can be found in the W3C web site [25].

- **Main classes:** Main classes in RDF Schema are *rdfs:Resource*, *rdf:Property*, and *rdfs:Class*. The class *rdfs:Resource* is the class of everything that is described by RDF. Therefore, all of the resources in RDF are instances of the class *rdfs:Resource*. The class *rdf:Property* is the class of all RDF properties and is itself an instance of the *rdfs:Class*. Concepts are defined using *rdfs:Class* in RDF Schema. Besides, the class *rdfs:Class* is the class of those resources in RDF that are RDF classes.

- **Main properties:** Main properties in RDF Schema are *rdfs:subClassOf*, *rdfs:subPropertyOf*, and *rdf:type*. All of them are instances of *rdf:Property*. The property *rdfs:subClassOf* is used to state hierarchy between classes, which means that, instances of one class are also instances of another one. The property *rdfs:subPropertyOf* is used to state hierarchy between properties, which means if a resource is related by one property it is also related by another one. The property *rdf:type* is used to identify that a resource is related to a class and it is an instance of the class.

- **Main constraints:** Main constraints in RDF Schema are *rdfs:domain* and *rdfs:range*. Both of them are instances of *rdf:Property*. The property *rdfs:domain* states that any resource with a given property is an instance of one or more specific class.

For instance, the triple *P rdfs:domain C* states that the subjects of triples are instances of class *C* if those triples' predicates are *P*. The property *rdfs:range* states that all allowed values of a property are instances of one or more specific classes. For instance, the triple *P rdfs:range C* states that the objects of triples are instances of class *C* if those triples' predicated are *P*.

2.7 OWL (Web Ontology Language)

This section presents a brief overview about the concepts of the OWL.

2.7.1 Overview

OWL is a language for defining Web ontologies [26]. OWL is the recommendation of W3C to make Web resources more processable for applications by adding information about the resources. OWL is used in cases that information in documents needs to be processed by applications, rather than only being displayed to humans.

OWL makes it is possible to represent the meaning of terms that are used in vocabularies and also relationships between those terms. Generally, the representation of the terms and identifying their relationships is called Ontology. Using additional vocabulary along with formal semantics, OWL provides more facilities than XML, RDF, and RDF Schema for expressing meaning and semantics. As a result, it has more abilities for representing machine interpretable contents. [27] [3] [28]

OWL is the revision of the earlier languages OIL (Ontology Inference Layer) [29] and DAML+OIL (DARPA Agent Markup Language) [30].

2.7.2 Sublanguages

OWL provides three sublanguages OWL Lite, OWL DL, and OWL Full which are designed for use of specific users and communities [3].

- **OWL Lite:** OWL Lite is designed for those users who need a classification hierarchy and simple constraints. For example, it supports cardinality constraints but only permits 0 or 1 for cardinality values. Quick migration from thesauri and taxonomies are possible using OWL Lite and because it has lower complexity than OWL DL, providing tools for supporting OWL Lite is simple.

- **OWL DL:** OWL DL is proper for those users who need maximum expressiveness with computational completeness and decidability. "Computational completeness" means that all deductions are computable and decidability means that all computations process will be finished in finite time. OWL DL contains all of the OWL constructs with some certain restrictions of usage.

- **OWL Full:** OWL Full is proper for those users who need maximum expressiveness and the syntactic freedom of RDF. However, there is no guarantee for computational completeness and not all conclusions are guaranteed to be computable.

There are some relations between these sublanguages in terms of legal expressions and valid conclusions. Each sublanguage is an extension of its predecessor and the following relations are true [3]:

- Each legal OWL Lite ontology is a legal ontology in OWL DL.
- Each valid OWL Lite conclusion is a valid conclusion in OWL DL.
- Each legal OWL DL ontology is a legal ontology in OWL Full.

- Each valid OWL DL conclusion is a valid conclusion in OWL Full.

Every OWL document including OWL Lite, OWL DL, and OWL Full is a RDF document. Also all RDF documents are OWL Full documents but only some of them will be legal OWL Lite or OWL DL documents.

2.7.3 Modeling Primitives

This section briefly reviews modeling primitives of OWL according to the W3C documents [27], [31]. OWL ontology is a RDF graph, which is consisting of some RDF triples. Like RDF graph, OWL ontology can be written in different formats and RDF/XML syntax is the popular one.

The built-in vocabulary in OWL is associated with a namespace called "owl" and comes from OWL namespace "http://www.w3.org/2002/07-/owl". MIME (Multi-purpose Internet Mail Extension) type of the OWL documents is "application/rdf+xml" and for file extension, either ".rdf" or ".owl" is recommended.

"Classes" in OWL are described through "class descriptions" and there are six types of them:

- **Class identifier:** describes a class through a "class name" which is represented as a URI reference. For instance, `<owl:Class rdf:ID="Person"/>` describes a class called Person but it does not tell much about the class Person. "Class axioms" are used to state necessary characteristics of a class. OWL contains three constructs for class axioms including rdfs:subClassOf, owl:equivalentClass, and owl:disjointWith. The rdfs:subClassOf construct states hierarchy between classes and owl:equivalentClass construct states that class extensions of two class descriptions are exactly same. The owl:disjointWith construct states that class

extensions of two class descriptions have no members in common. The following code is an example of using owl:disjointWith:

```
<owl:Class rdf:about="#Man">
  <owl:disjointWith rdf:resource="#Woman"/>
</owl:Class>
```

- **Enumeration**: describes a class by an exhaustive collection of individuals that form the instances of that class. As an example the following syntax defines a specific class of colors:

```
<owl:Class>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Blue"/>
    <owl:Thing rdf:about="#Red"/>
    <owl:Thing rdf:about="#Green"/>
  </owl:oneOf>
</owl:Class>
```

- **Property restriction**: describes an anonymous class of all individuals that fulfill the property restriction. OWL has two types of property restrictions including "value constraints" and "cardinality constraints". A value constraint puts constraints on the range of property while cardinality constraint puts constraints on the number of values that a property can take. There are different types of value constraints and cardinality constraints in OWL. For example, by using value constraint "allValueFrom" the following code describes an anonymous OWL class of all individuals in which the property "hasParent" can only have values from class "Person":

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:allValuesFrom rdf:resource="#Person" />
</owl:Restriction>
```

- **Intersection**: describes a class as an intersection of two or more class descriptions. For example, the following code describe a

class, which is an intersection of two class descriptions, both have two containing two individuals. The result will be a class with one individual, “Blue”.

```
<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Blue" />
        <owl:Thing rdf:about="#Red" />
      </owl:oneOf>
    </owl:Class>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Blue" />
        <owl:Thing rdf:about="#Green" />
      </owl:oneOf>
    </owl:Class>
  </owl:intersectionOf>
</owl:Class>
```

- **Union**: describes an anonymous class as a union of two or more class descriptions. As an example the following code describes an anonymous class with three individuals including “Blue”, “Red”, and “Green” using union descriptions:

```
<owl:Class>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Blue" />
        <owl:Thing rdf:about="#Red" />
      </owl:oneOf>
    </owl:Class>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Blue" />
        <owl:Thing rdf:about="#Green" />
      </owl:oneOf>
```

```
    </owl:Class>
  </owl:unionOf>
</owl:Class>
```

- **Complement**: describes a class containing those individuals that do not belong to another class. As an example, the expression "Female" can be expressed as the following code:

```
<owl:Class>
  <owl:complementOf>
    <owl:Class rdf:about="#Male"/>
  </owl:complementOf>
</owl:Class>
```

"Properties" in OWL are available in two main categories including "object properties" and "datatype properties". Object properties are used to link individuals to individuals and datatype properties are used to link individuals to data values. For example, `<owl:ObjectProperty rdf:ID="hasParent"/>` defines property "hasParent" as an object property.

There are also property axioms in OWL for defining additional characteristics of properties. OWL contains different constructs for property axioms including RDF Schema constructs, property relational constructs, global cardinality constraints, and logical property characteristics. RDF Schema constructs consist of rdfs:subPropertyOf, rdfs:domain, and rdfs:range are described in section 2.3.2.

Property relational constructs consist of owl:equivalentProperty and owl:inverseOf. The owl:equivalentProperty construct states that the property extensions of two properties are same. The owl:inverseOf construct states the inverse relation between two properties. For example, the following code defines inverse relation between properties "hasChild" and "hasParent":

```
<owl:ObjectProperty rdf:ID="hasChild">
  <owl:inverseOf rdf:resource="#hasParent"/>
</owl:ObjectProperty>
```

Global cardinality constraint constructs consist of owl:FunctionalProperty and owl:InverseFunctionalProperty. The owl:FunctionalProperty construct states that a property can only have one value for each instance. For example, the following axiom states that the property "idNumber" is functional because every person can have only one id number:

```
<owl:ObjectProperty rdf:ID="idNumber">
  <rdf:type    rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Person" />
  <rdfs:range  rdf:resource="#ID" />
</owl:ObjectProperty>
```

The owl:InverseFunctionalProperty construct is used to state that a property is inverse functional. It means that the object of the property statement can uniquely determine the subject.

Logical property characteristics constructs consist of owl:TransitiveProperty and owl:SymmetricProperty. The owl:TransitiveProperty is used to state that a property is transitive. It means that when pairs (x,y) and (y,z) are instances of the property, then the pair (x,z) is an instance of that property. The owl:SymmetricProperty is used to state that a property is symmetric. It means that when a pair (x,y) is an instance of the property, then (y,x) is also an instance of that property.

"Individuals" in OWL are defined by individual axioms. There are two types of individual axioms including class membership and individual identity. An individual can be introduced by declaring it as a member of a class. For example, the axiom `<Color rdf:ID="Blue"/>` indicates that individual "Blue" is a member of class "Color".

There are three individual identity constructs in OWL including owl:sameAs, owl:differentFrom, and owl:AllDifferent which are used to

state some facts about the individual's identity. The owl:sameAs construct states that two different URIs refer to the same individual. The owl:differentFrom construct states that two URIs refer to the different individuals. For example, the following code states that there are two different colors:

```
<Color rdf:ID="Blue"/>
<Color rdf:ID="Red">
  <owl:differentFrom rdf:resource="#Blue"/>
</Color>
```

Finally owl:AllDifferent construct states that all declared individuals in the list are all different. As an example, the following code declares that all three URIs are different colors:

```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <Color rdf:about="#Blue"/>
    <Color rdf:about="#Red"/>
    <Color rdf:about="#Green"/>
  </owl:distinctMembers>
</owl:AllDifferent>
```

2.8 RDF Query Languages

RDF query language is used to get information out of a knowledgebase and manipulate stored data in RDF format. It allows end users and developers to write desired queries and consume the query results across broad range of information on the Web. Several languages have been proposed for querying RDF documents and SPARQL is introduced as a standard query language for RDF documents by W3C.

2.8.1 Various Query Languages

Several query languages such as RQL (RDF Query Language), SeRQL (Sesame RDF Query Language), SquishQL, RDFPath, Versa, TRIPLE, DAML+OIL Query Language, RDQL, RDFQL, N3, iTQL, RStar, SPARQL, and so on have been introduced for RDF documents.

All of the mentioned query languages were intended to provide a proper query language for RDF documents. Some of them including RQL, SeRQL, TRIPLE, RDQL, N3, and Versa have been described and compared in "A Comparison of RDF Query Languages", reference [32].

2.8.2 SPARQL

SPARQL is the standard RDF query language and data access protocol introduced by W3C for easy accessing to RDF documents in Semantic Web. It is defined in connection with RDF data model and works with any data source that can be expressed by RDF. The SPARQL query language provides syntax and semantics for getting information from RDF graphs. It provides some facilities to extract information in various forms such as URIs, blank nodes, and plain and typed literals. It also has facilities to extract RDF sub graphs and construct new RDF graphs using information in the queries [33].

Matching graph patterns is the basis of SPARQL query language. The simplest basic graph pattern can be a triple pattern like RDF triple with variables instead of subject, predicate, and object. These graph patterns can be used as basic patterns for RDF graphs in order to retrieve query results. There is no inference in the SPARQL query language and it only queries the information in RDF graphs.

As an example, consider the following RDF data, which is presented in section 2.2.2:

```
<rdf:RDF
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:dc="http://purl.org/dc/elements/1.1/"
       xmlns:foaf="http://xmlns.com/foaf/0.1/" >

<rdf:Description rdf:about="http://www.bogus.com/index.html">
  <dc:creator rdf:resource="http://www.bogus.com/staffid/10"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.bogus.com/staffid/10">
  <foaf:name>Bob</foaf:name>
</rdf:Description>

</rdf:RDF>
```

The following example shows a simple SPARQL query to find the "name" of the staff member 10 from the information in the above given RDF graph.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
{
   <http://www.bogus.com/staffid/10> foaf:name ?name .
}
```

In the above code, PREFIX foaf defines namespace of vocabulary http://xmlns.com/foaf/0.1/, SELECT clause of the query identifies variable "name" which is the query result, and WHERE clause of the query contains one triple pattern. The result of query based on the mentioned RDF graph will be:

name
"Bob"

A complete description of the SPARQL query language is available on W3C site [33].

2.9 Implemented applications

Some of the implemented applications based on Semantic Web are briefly explained in this section as a sample of Semantic Web applications.

- **Friend of a Friend (FoaF):** Friend of a Friend (FoaF) project is one of the popular implemented applications of Semantic Web. It is about describing people, relationships among them and the things they have created in the form of a machine-readable web. FOAF is implemented on the basis of RDF and defined using OWL in order to share data between different environments. [13]

- **BigBloZoo:** BigBlogZoo is a Semantic Web Browser in which around 70,000 News feeds and Blogs, which are called channels, have been categorized using the DMOZ[1] Schema. This information is in machine-readable XML format. The BigBlogZoo allows a web user to search and browse those channels and save the results as channels. [14]

- **Piggy Bank:** Piggy Bank is a new free plug-in and an extension to the Firefox web browser, which makes it as a Semantic Web browser. Existing information and web scripts on the web are extracted and translated into RDF information and stored on the web user's local computer using Piggy Bank. Therefore, this information can be used independently in other contexts in more useful ways. [8] [15]

[1] Another name for the Open Directory Project (ODP): the largest human edited directory of the Web which is maintained by a community of volunteer editors. For more information see: http://www.dmoz.org/

Chapter 3

Method

This chapter presents the vision document of the project. It is written based on the vision template of Rational Unified Process (RUP) [34]. The Rational Unified Process is an iterative framework for software development process, which is created by the Rational Software Corporation [35], [36].

The goal of this chapter is to collect and define high-level features and needs of the Ultimate Ontology Management System (UltimateOMS). This section focuses only on the required capabilities of the stakeholders and users, and why these needs exist. The details of how UltimateOMS fulfills these needs are described in the use case specifications in appendix B.

3.1 Positioning

3.1.1 Business Opportunity

This software will be a new ontology management system based on Jena, which is an open source java framework for building Semantic Web applications. The new tool will enable users to create and manage their semantic data (ontology components) using the web based user interface.

The new tool will bring necessary functions for managing semantic data in one place thus making it much easier for users to create and manage ontology components.

3.1.2 Problem Statement

The problem of	The lack of proper tool for creating and managing of semantic data (ontology components)
Affects	Users dealing with semantic data
The impact of which is	A slow and costly timely process to create and manage semantic data
A successful solution would be	Developing a new application in which all necessary functions are provided in one place in order to facilitate creating and managing of semantic data

3.1.3 Product Position Statement

For	Users dealing with semantic data
Who	Want to create and manage semantic data (ontology components)
"Ultimate Ontology Management System	Is a software product

(UltimateOMS) ”	
That	Facilitate creating and managing of semantic data by providing all necessary functions in one place
Unlike	The current exciting tools for creating and managing semantic data, which do not provide all required features for users dealing with semantic data
Our product	Provides an ultimate ontology management system with all necessary functions in one place in order to facilitate creating and managing semantic data (ontology components)

3.2 Stakeholder and User Descriptions

This section describes the profile of the stakeholders and users of the Ultimate Ontology Management System and the key problems that should be addressed by the proposed solution. There are two types of users of UltimateOMS: Administrator, and Users.

3.2.1 Market Demographics

The target market for this system includes all users who deal with semantic data and intend to create, manage, use, and even share semantic data in the Web.

3.2.2 Stakeholder Summary

The following summary list, presents all identified stakeholders in the system.

Name	Description	Responsibilities
Administrator	Administrators of the system	Ensures that the system will meet the needs of the ontology management system, who has to install and manage the application, maintain databases, and modify system configuration
User	End users of the system	Represents the interests of the users and ensures that the system will meet the needs of system end users

3.2.3 User Summary

The following summary list, presents all identified users in the system.

Name	Description	Stakeholder
Administrator	Installs and manages application, maintains databases, and modifies system configuration	self-represented
User	Creates and manages semantic data (ontology components)	self-represented

3.2.4 User Environment

Users will require a web browser such as "Mozilla Firefox" and "Internet Explorer" in order to interact with system for creating and managing their semantic data.

3.2.5 Stakeholder Profiles

The following lists describe each identified stakeholder in the system.

- **Administrator:**

Description	Individuals who are administrators of the system
Type	The administrator is an expert user who knows detail information about the application
Responsibilities	The administrator is responsible for reviewing system requirements
Success Criteria	Ability to maintain the application
Involvement	Requirements reviewer
Comments / Issues	None

- **User:**

Description	Individual who is an end user of the system dealing with semantic data
Type	The User is a primary and end user of the system
Responsibilities	The User ensures that the system will be acceptable to users in terms of provided features and ease of use
Success Criteria	Ability to work easily with system and create and manage semantic data
Involvement	Provides reviews of preliminary version of software in order to fix problems
Comments / Issues	None

3.2.6 User Profiles

User profiles are covered under the previous section.

3.2.7 Key Stakeholder or User Needs

The following list describes the main concerns and needs of the system's stakeholders and users.

Need	Priority	Concerns	Current Solution	Proposed Solutions
Integrated Ontology Management System	High	Creating and managing of semantic data is a slow, costly and timely process	Currently users must use different tools in order to create and manage their semantic data	Users would like to have integrated application in which all necessary functions are provided in one place in order to facilitate creating and managing of semantic data
Web based user interface	Medium	Maintaining a new version of software in client side is costly	Using web based application technology	Using web based application technology
Ease of use	High	Ability to provide navigations in user interface	Using standard navigations in web interface	Designing a new user friendly web interface
Storing semantic data in different databases	High	Ability to store semantic data in different databases	Storing data in different databases using JDBC in Java applications	Storing data in different databases using JDBC in Java applications

Visualization of semantic data	High	Ability to make graph of the generated semantic data	Currently users must different types of tools to create semantic data and generate graph	Providing graph facilities seamless in application by using Graphviz as a graph generator
Scalability	Medium	Ability to store hug number of semantic data records in database	Using a scalable database	Supporting different types of databases for different types of user requirements
Security	Medium	User data	Using appropriate security mechanism provided in web technology and database	Using database security mechanisms, like identifying system users as database users or any identity management system that the database supports such as using LDAP directories

3.3 Product Overview

This section delivers a high-level view of UltimateOMS capabilities, interfaces to the external Semantic Web Framework and supported back-end databases, and Graph Generator System.

3.3.1 Product Perspective

UltimateOMS leverages from the existing Semantic Web Framework, Jena and its supported back-end databases, and graph generator system as shown graphically in the figure 3.1.

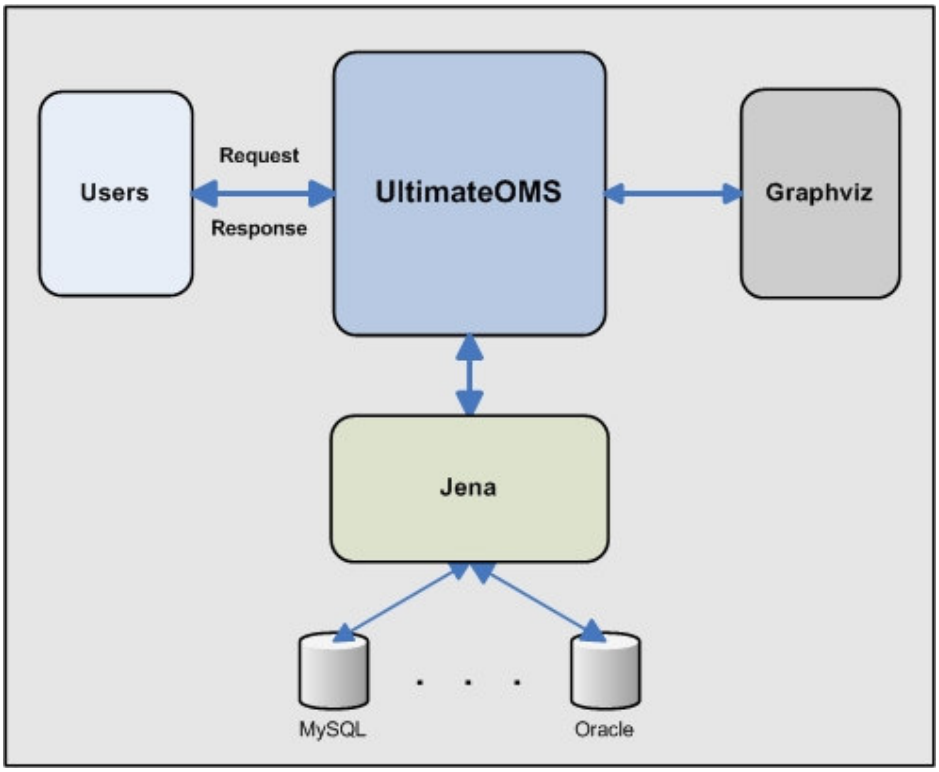

Figure 3.1: UltimateOMS high-level diagram

UltimateOMS will consist of server components, which reside on UltimateOMS server. The server components use the Jena APIs and interact with Graphviz for generating graph, which are both resided on

the same server as UltimateOMS resides on. However, databases can be installed on either different servers or the same server, which is used for UltimateOMS.

There is no specific client component in UltimateOMS because it is a web based application. The users may access UltimateOMS from their PC either through the LAN or through Internet using a web browser. A valid username and password is required in order to access UltimateOMS.

3.3.2 Summary of Capabilities

The following table identifies the main capabilities of UltimateOMS in terms of benefits and features.

Customer Benefit	Supporting Features
Easy and convenient access	Users may access the ontology management system from anywhere through Internet using a web browser
Secure access	A valid username and password is required in order to gain access to UltimateOMS
Easy to use	A user friendly web based user interface makes it very easy to users for managing Semantic data

Easy to manage and manipulate Semantic data	UltimateOMS provides the necessary functions in one place in order to facilitate creating, manipulating, and managing Semantic data (ontology components)
Broad choice for backend system	Users may store and manage their semantic data in different types of databases
Visualization graph of Semantic data	UltimateOMS provides graph facilities seamless in application by using Graphviz as a graph generator
Querying Semantic data and exporting the result	Users may query Semantic data using querying facilities provided by UltimateOMS and export the query results
Inferring Semantic data	Users may infer Semantic data using inferring facilities provide by UltimateOMS

Table 3.1: Capabilities of UltimateOMS

3.3.3 Assumptions and Dependencies

The following outlined assumptions and dependencies are related to the capabilities of UltimateOMS:

- It is assumed that a correct version of Java platform, Standard Edition, J2SE 1.5 or higher is installed on UltimateOMS server.

- It is assumed that an application server such as Tomcat, JBoss, or Oracle Internet Application Server is installed on UltimateOMS server.

- The current version of Jena, at the moment Jena-2.5.2, is included in UltimateOMS package.

- It is assumed that a proper version of Graphviz is installed on UltimateOMS server.

- It is assumed that all required databases are installed and configured on either different servers or the same server, which is used for UltimateOMS.

3.4 Product Features

This section defines and briefly describes the features of UltimateOMS. Features are the high-level capabilities of the system that are essential to give benefits to the users. Each feature will be described in detail in the use case model. For more information regarding use case model see appendix B.

- ### Modify UltimateOMS System Configuration

  The system shall allow the administrator to modify system configuration file. It includes some parameters related to databases and graph. The administrator can modify the configuration file for the following actions:

  - Adding new database
  - Modifying database parameters
  - Removing database
  - Modifying graph parameters

- Logon

  Users shall provide a valid username and password and specify an available database for access to UltimateOMS. Administrators will assign usernames and passwords to the users. These usernames and passwords should match the database usernames/passwords.

- Show and Search Models

  The system shall display all available users' models, which already exist in the connected database, as a list to the users. The system shall also allow the users to search for a specific list of models stored in the connected database by typing some part of desired model's name. The system shall display all the filtered models as a list to the user in which model's names contain the user typed part.

- Create, Import, and Upload Model

  The system shall allow the users to create an empty model in connected database, import a model in different formats from anywhere in Internet into the connected database, and upload a model in different formats from local file system into the connected database by providing all necessary information. The new model shall be displayed in *Show Model* list.

- Select a Default Model

  The system shall allow the users to select a default model from the list of existing models in database in order to work with it.

- Export Model

  The system shall allow the users to export the default model from the connected database in different formats into local file system by providing all necessary information. The users shall be asked to specify a local directory path in which the exported model will be saved.

- Show and Export Model Graph

  The system shall enable the users to view graph of the default model by parsing the default model and sending a proper file to Graphviz. The generated Graphviz file shall be displayed to the users using a java applet application. The system shall also allow the users to export the generated model graph in different formats into local file system by providing all necessary information. The users shall be asked to specify a local directory path in which the exported graph of model will be saved.

- Query Model and Export Query Results

  The system shall enable the users to query the model from the connected database by providing desired SPARQL query. The system shall display all the query results as a list to the user. The system shall also allow the users to export the query results in different formats into local file system by providing all necessary information. The users shall be asked to specify a local directory path in which the exported query results of model will be saved.

- Infer Model

  The system shall enable the users to infer the model from the connected database using different available types of "Reasoner" or "Generic Rule Reasoner" by providing all necessary

information. The inferred results shall be saved in the connected database as a new model with the specified name.

- Show Model Statistics

  The system shall enable the users to see all statistics information of the model.

- Check Model Consistency

  The system shall enable the users to check the model consistency in the connected database by providing all necessary information. The generated report shall be displayed to the users.

- Delete Model

  The system shall enable the users to delete a selected model from the connected database.

- Show and Search Triples

  The system shall display all the existing triples of the model as a list to the users. The system shall also allow the users to search for a specific list of triples of model by typing some part of the desired triple's subject name, predicate name, object name, or all of them. The system shall display all of the filtered triples of the model as a list to the user.

- Browse Triples

  The system shall enable the users to browse triples of a selected resource in model. The system shall display all the filtered triples of the model as three different lists to the user in which triples'

subject name, predicate name, or object name is equal to the selected resource.

- Select Triple

  The system shall allow the users to select a triple from the list of existing triples in model in order to deal with it.

- Create, Edit, and Delete Triple

  The system shall allow the users to create a triple for model by providing all necessary information, edit a selected triple of model by modifying all necessary information, and delete a selected triple from model.

- Show and Search Classes

  The system shall display all the existing classes of the model as a list to the users. The system shall also allow the users to search for a specific list of classes of model by typing some part of the desired classes' name. The system shall display all the filtered classes of the model as a list to the user in which classes' names contain the user typed part.

- Select Class

  The system shall allow the users to select a class from the list of existing classes in model in order to deal with it.

- Create, Edit, and Delete Class

  The system shall allow the users to create a class for model by providing all necessary information, edit a selected class of model

by modifying all necessary information, and delete a selected class from the model.

- Create Instance for Class

  The system shall allow the users to create an instance for a selected class by providing all necessary information.

- Show Class Details

  The system shall enable the users to see all categorized detail information of the selected class.

- Show Class Triples

  The system shall enable the users to see all the existing triples of the selected class as a list.

- Show and Search Properties

  The system shall display all the existing properties of the model as a list to the users. The system shall also allow the users to search for a specific list of properties of the model by typing some part of desired properties' name. The system shall display all the filtered properties of the model as a list to the user in which properties' names contain the user typed part

- Select Property

  The system shall allow the users to select a property from the list of existing properties in model in order to deal with it.

- Create, Edit, and Delete Property

  The system shall allow the users to create a property for model by providing all necessary information, edit a selected property of model by modifying all necessary information, and delete a selected property from model.

- Show Property Details

  The system shall enable the users to see all categorized detail information of the selected property.

- Show Property Triples

  The system shall enable the users to see all the existing triples of the selected property as a list.

- Show and Search Individuals

  The system shall display all the existing individuals of the model as a list to the users. The system shall also allow the users to search for a specific list of individuals of model by typing some part of desired individuals' name. The system shall display all the filtered individuals of the model as a list to the user in which individuals' names contain the user typed part

- Select Individual

  The system shall allow the users to select an individual from the list of existing individuals in model in order to deal with it.

- Edit and Delete Individual

  The system shall allow the users to edit a selected individual of model by modifying all necessary information and delete a selected individual from model.

- Show Individual Details

  The system shall enable the users to see all categorized detail information of the selected individual.

- Show Individual Triples

The system shall enable the users to see all the existing triples of the selected individual as a list.

3.5 Constraints

In addition to the mentioned assumptions and dependencies in Section 3.4.3, the following constraints apply to UltimateOMS:

- The system shall not have any dependency to any specific operating system.

- The system reliability and performance in terms of response time are limited to the reliability and performance of the Jena Semantic Web Framework.

Chapter 4

System Design

This chapter reviews the technologies and concepts related to the architecture of the application, and finally presents the architecture of the proposed solution for implementing Ultimate Ontology Management System (UltimateOMS).

4.1 JSP Technology and Java Servlets

JavaServer Pages (JSP) technology makes Web developers capable of rapidly developing and maintaining platform independent web based applications. JSP technology separates the business logic from user interface design, enabling user interface designers to change page design and layout without changing the business logic. The logic, which generates the page contents, is encapsulated in the specific tags in JSP files. The application business logics can reside in the server in the form of Java Beans components, which are accessible by JSP pages tags.

Java Servlets are server side and platform independent components that can be used to extend Web server capabilities with minimum maintenance and overhead.

JSP technology and Java Servlets can be used together in order to develop platform independent applications which have enhanced performance, separated business logic and user interface design, and ability to extend into enterprise applications. [37]

A simplified architecture of JSP and Java Servlets technologies [44] are shown in figure 4.1.

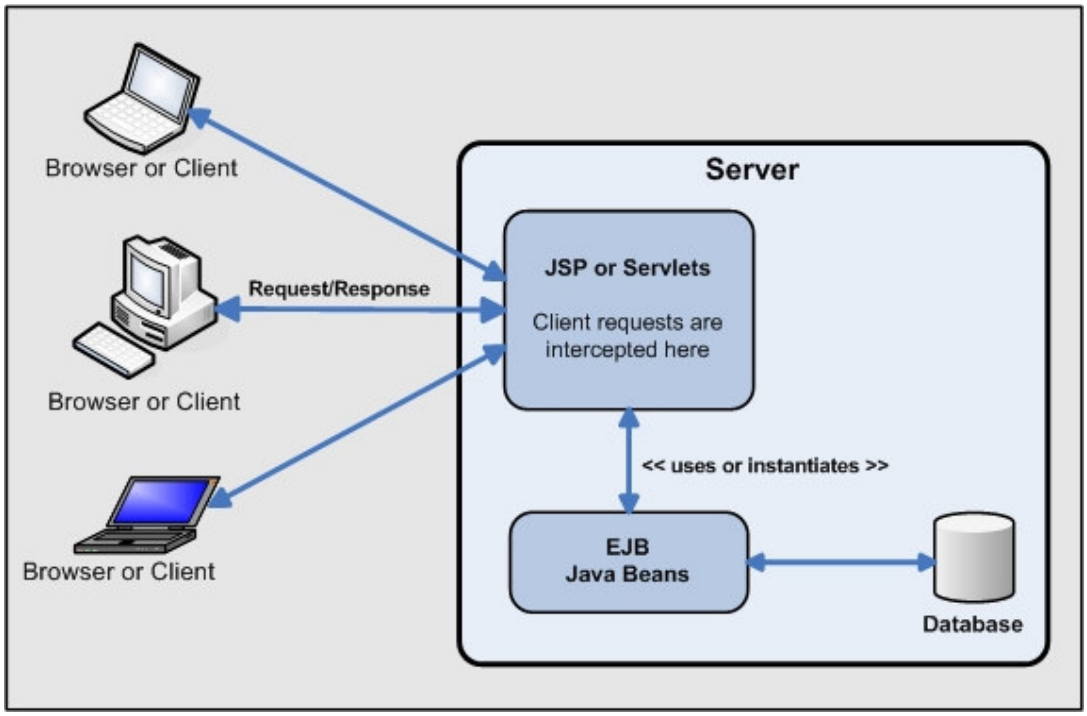

Figure 4.1: JSP and Servlets, simplified architecture

4.2 Semantic Web Framework: Jena

Jena is an open source Java framework for building Semantic Web applications developed by the Hewlett-Packard Company. Jena framework provides Application Program Interfaces (API) for RDF, RDFS, OWL, and SPARQL and also includes a rule based inference engine [38].

Jena has a simple abstraction of the RDF graph as an interface, which facilitates implementations of in-memory, database-backed, and inferred graphs. Persistence for RDF graphs is implemented by Jena database subsystem using back-end relational databases and through JDBC connections [19]. Therefore, using provided Jena model interfaces, users can store RDF graphs in different supported relational databases.

The latest version of Jena, Jena 2.5.2, supports MySQL, HSQLDB, PostgreSQL, Oracle, Apache Derby, and Microsoft SQL Server databases [39].

Jena2 is the second generation of the Jena that complies with the revised RDF specification. As shown in Graph Layer in figures 4.2, the RDF graph is the heart of the Jena2 [5]. The Graph Layer design, which is based on the RDF Abstract Syntax [17], is minimal and implementation is done as much as possible in other layers. Therefore, it is possible to have a range of different implementations for Graph Layer. Jena2 provides different types of implementations for Graph Layer such as in-memory and persistence triples stores.

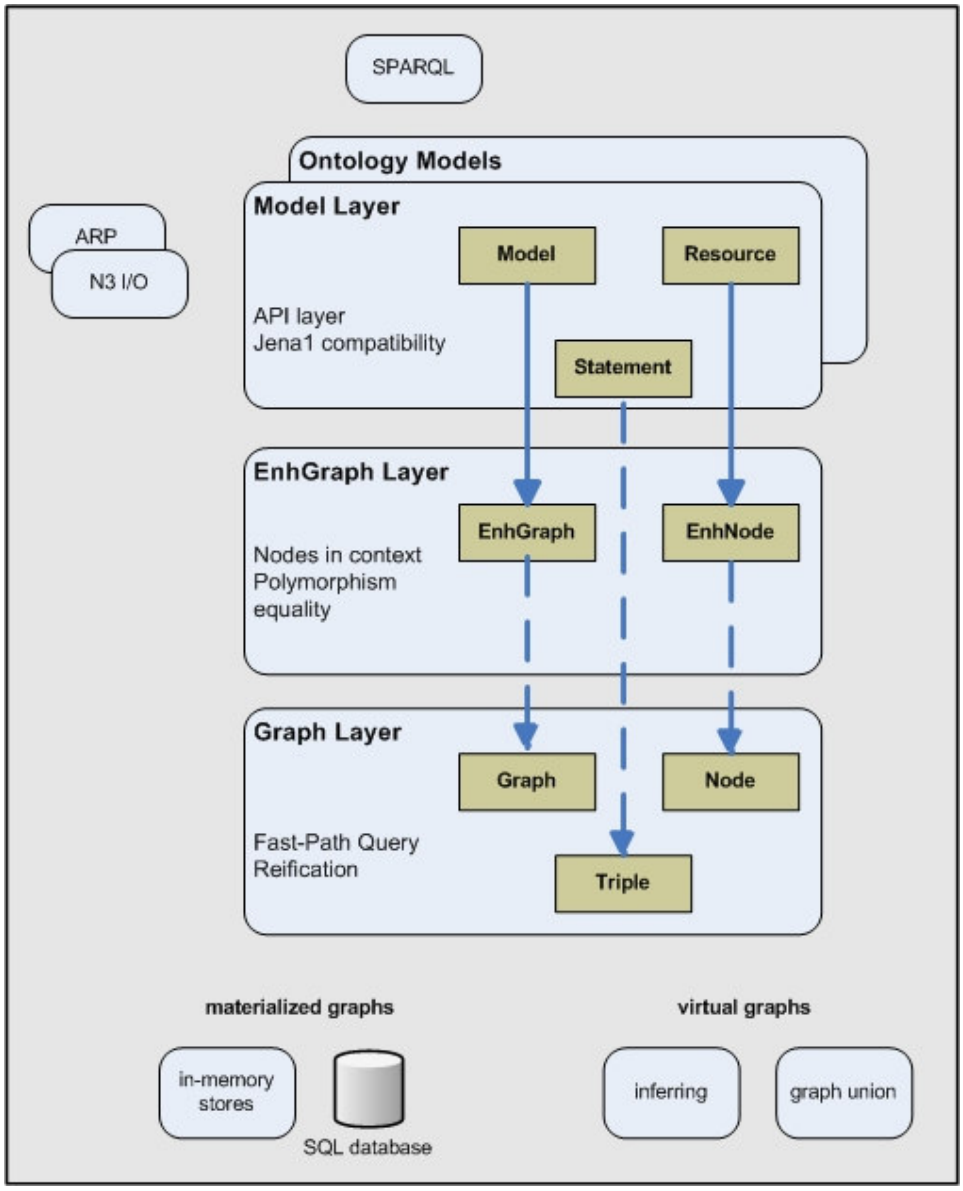

Figure 4.2: Jena2 architecture

EnhGraph Layer is an intermediate layer between both the Model and Ontology Layers and the Graph Layer. The EnhGraph provides multiple simultaneous views of graphs and nodes, which can be used by Model and Ontology APIs. In Jena, all states are kept within the Graph and the other presentation layers are stateless. [40]

Due to compatibility with Jena1, I/O is done in the Model Layer. Model API, which is the primary abstraction of the RDF Graph in Jena1, is

maintained in Jena2. The Model API provides a complete set of methods for applications in order to operate on both the Graph and Nodes within graphs. [40]

Jena is a leading Semantic Web programmers' toolkit, which is in widespread use [41]. It is available for download from SourceForge[2]. In order to keep compatibility with ongoing W3C standards and also provide much richer APIs, Jena still is under development. Support is provided through the Jena-dev mailing[3] list on Yahoo.

4.3 Databases

Considering the fact that users usually deal with large amount of semantic data, it is necessary to provide a mechanism for users in order to store and retrieve this data in an efficient way. The good choice is to use the conventional RDBMS databases. To do this efficiently, a very good option is to choose a Semantic Web framework, which provides not only the APIs for storing and retrieving Semantic data but also supports wide ranges of relational databases.

Jena has the APIs for storing and retrieving Semantic data and also supports persistent storage of RDF data in a wide range of relational databases. [39] Jena provides many APIs and java interfaces such as *Model* and *Query* to access and manipulate stored RDF data in databases. Moreover, for consistency reasons, applications are not allowed to access the stored data in databases directly and Jena provides all necessary APIs for dealing with stored data instead of accessing it directly.

---

[2] SourceForge.net is a centralized location for open source software developers to control and manage software development.

[3] Mailing list of Jena-dev is: jena-dev@groups.yahoo.com

4.3.1 Denormalized Schema

Storing triples is the widely used scheme for storing RDF statement in relational databases. Jena uses this approach to store each RDF statement, including subject, predicate, and object, as a single row in the *Statement Table*, which has three columns corresponding to them. In addition to those three columns, the *Statement Table* has another column indicating if the object of that statement is a literal or a URI.

Jena1 uses normalized schema approach, which uses less storage space in databases. In normalized approach, the literals and resources of all statements are stored in *Literal Table* and *Resource Table*. Therefore, *Statement Table* stores references to the values of resources and literals instead of storing their value directly. As a result, in this approach less space is used; because the literals and URI resource values are only stored once but used several times. However because retrieving a statement requires a join on three different tables, the performance will be affected as the number of statements increase. [19] [40] [42]

In order to have efficient retrieval, Jena2 uses demoralized schema approach, which is a hybrid approach of normalized and standard triple store. Using this approach in Jena2, short literals and URI resources will be stored directly in the *Statement Table*. However, long literals and URI resource are only stored once and only the references will be stored in the *Statement Table* similar to the approach in Jena1.

Jena2's approach uses more storage space than Jena1, but it has better response time and it is better in terms of space-time trade off. The threshold length for short value versus long value is 256 by default, but it is configurable and it is possible to change it to adjust the space-time trade off in different type of applications.

4.3.2 Tables

This section presents briefly the database tables, which are used in Jena2. More information regarding to the details of table designs is available in "Jena2 Database Interface - Database Layout", reference [42].

There are two different types of tables in Jena2: *Statement Tables* and *System Tables*.

- Statement Tables

Jena2 uses *Statement Tables* in order to store asserted statements and reified statements. There are two different tables as following:

- **Asserted Statement Table**

  This table holds asserted statements for one or more graphs. By default, statements of each graph are stored in its own statement table, which name has the form *Jena_GiTj_Stmt*. In this form *i* is graph identifier and *j* is table counter for graph.

- **Reified Statement Table**

  This table holds reified statements for one or more graphs. By default, reified statements of each graph are stored in its own reified statement table, which name has the form *Jena_GiTj_Reif*. In this form *i* is graph identifier and *j* is table counter for graph.

- System Tables

Jena2 uses *System Tables* in order to store metadata and long values for literals and resources. There are six different tables as following:

- **System Statement Table**

  This table, *Jena_Sys_Stmt*, holds system metadata such as configuration parameters and table names for graphs for the Jena2.

- **Long Literals Table**

  This table, *Jena_Long_Lit*, holds the literals that are long to store directly in statement tables.

- **Long Resources Table**

  This table, *Jena_Long_URI*, holds the resources that are too long to store directly in statement tables.

- **Prefixes Table**

  This table, *Jena_Prefix*, holds common URI prefixes in order to minimize used space in database.

- **Graph Table**

  This table, *Jena_Graph*, holds the name and unique identifier for user graphs.

- **Lock Table**

  This table, *Jena_Mutex*, holds some information, which are used internally in Jena2 in order to implement some critical sections.

4.3.3 Supported Databases

Currently the latest version of Jena, Jena 2.5.2, supports the databases which are listed in the table 4.1. The table also lists the JDBC drivers, which are compatible with Jena2. [39]

Database Engine	JDBC Driver
HSQLDB 1.8.0	
MySQL 4.1.11 MySQL 5.0.18	JDBC driver versions: 3.0, 3.1, 5.0
PostgreSQL 7.3 PostgreSQL 8.0	JDBC driver 7.3 JDBC driver 8.0
Apache Derby 10.1	
Oracle 10 XE	Oracle ojdbc14 driver (thin driver) 10.2.0.2
Oracle 9i Release 2	Oracle ojdbc14 driver (thin driver) 10.2.0.2
Oracle 10g Release 2	Oracle ojdbc14 driver (thin driver) 10.2.0.2
Microsoft SQL Server 2005 Express SP1	Microsoft SQL Server 2005 JDBC Driver
Microsoft SQL Server 2000 Microsoft SQL Server Desktop Edition	Microsoft SQL Server 2005 JDBC Driver jTDS version 1.2

Table 4.1: Supported database engines and JDBC drivers by Jena2

4.4 Graph Generator: Graphviz

Visual representation of RDF data is the easiest way for users in order to understand the structure of RDF data. To fulfill this goal, visual graph of RDF data has to be generated using a graph visualization software.

Graphviz (Graph Visualization Software), initiated by AT&T Research Labs, is an open source software for generating visual graphs. Graphviz can generate diagrams by taking descriptions of graphs in a simple text file. It also provides very useful features for diagrams such as colors and

fonts and also it can save them in different formats such as PNG, SVG and Postscript. [43]

Graphviz has several graph layout programs such as “dot”, “neato and fdp”, “twopi”, and “circo” in order to generate different types of diagrams. There are also some different types of viewers such as “dotty”, “tcldot”, “WebDot”, “Grappa”, “ZGRViewer”, and “Mac OS X graphviz” for different types of needs and environments. The latest version of Graphviz for different environments are available to download from *www.graphviz.org*.

4.5 UltimateOMS Architecture

The following diagram (see figure 4.3) shows the proposed architecture of UltimateOMS in order to fulfill user needs presented in chapter 3.

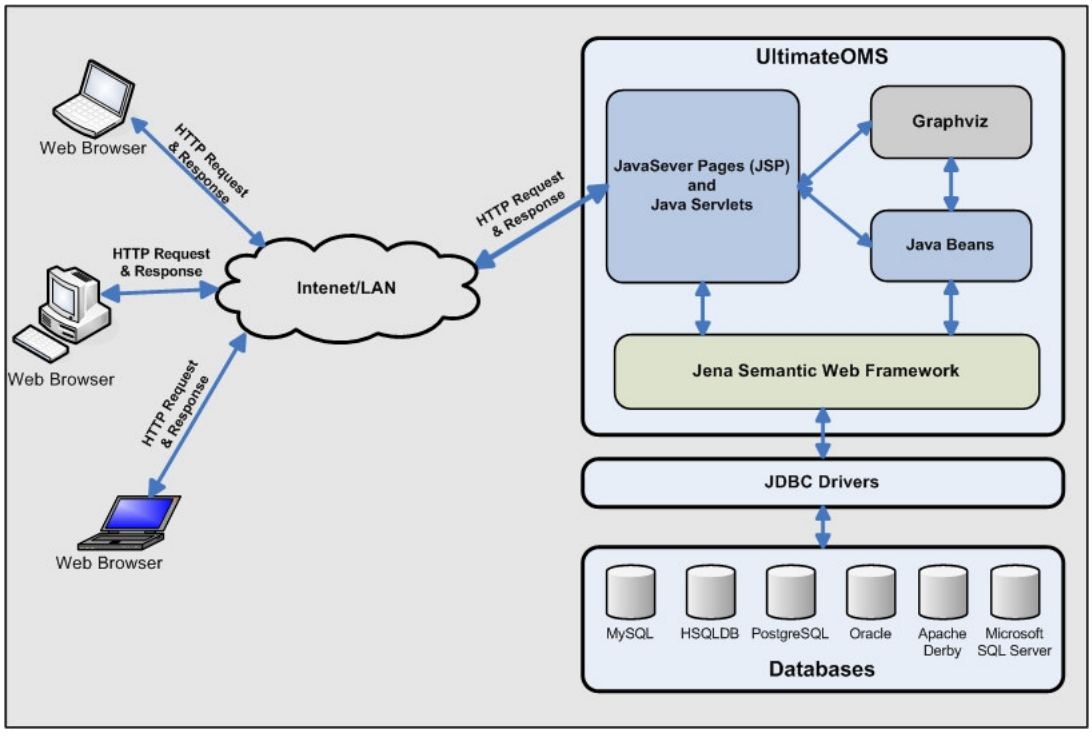

Figure 4.3: Architecture of UltimateOMS

4.5.1 JSP and Java Servlets framework

As shown in the architecture in Figure 4.3, the JSP and Java Servlets technologies are used together in order to develop UltimateOMS. Both JSP and Java Servlets are server side components that reside on an Application Server such as Tomcat, JBoss, Oracle Application Server.

Users' requests are collected by JSP and Java Servlets using HTTP Requests. The proper business logic, depending on the type of the request, will be executed using either Java Beans or Java Servlets. The operation results will be sent to users with HTTP Responses using JSP pages. Those JSP pages are the presentation layer and contain HTML tags and dynamic contents.

Because UltimateOMS is a web based application, there is no specific client component for clients. The users may access UltimateOMS from their PC either through the LAN or through Internet using a web browser. However, a valid username and password is required in order to access UltimateOMS.

Interaction between users and UltimateOMS will be carried out through HTTP protocol in order to send users requests and receive responses.

4.5.2 Jena

UltimateOMS leverages from Jena Semantic Web framework, in order to enable users with creating, manipulating, and managing Semantic data. Jena resides on the same server as UltimateOMS resides on.

Depending on users' requests and business logics, Jena APIs will be used by Java Beans, Java Servlets, and JSP pages in order to fulfill users' demands on Semantic data management including create, store, retrieve, manipulate, and more.

4.5.3 Databases

Databases are used by Jena Semantic Web framework in order to store and retrieve Semantic data. Jena supports wide range of databases such as MySQL, HSQLDB, PostgreSQL, Oracle, Apache Derby, and Microsoft SQL Server, as mentioned in the Section 4.3.3, through JDBC drivers. Required databases can be installed and configured on either different servers or the same server as UltimateOMS.

4.5.4 Graphviz

In order to generate visual graph of the Semantic data, UltimateOMS will interface with Graphviz. Graphviz resides on the same server as UltimateOMS. Both Graphviz and UltimateOMS use a shared area in the server in order to keep temporary files, which are used to communicate with each other. In our case, the shared area is a specific directory, which is accessible by both Graphviz and UltimateOMS.

Depending on users' requests for displaying or exporting graphs of Semantic data, Java Beans and Java Servlets will interact with the layout program of Graphviz, which is "dot" in our case, in order to generate the visual graph.

In case of displaying the generated graph in web browsers, UltimateOMS will send the generated graph to users through a Java applet application. In this case, web browsers in client side have to be Java enabled in order to load the Java applet and display the graph.

Chapter 5

Implementation

In order to fulfill the specified user needs in chapter 3, a software prototype has been developed. This chapter presents detail information about the implementation of UltimateOMS whose architecture is shown in figure 4.3.

5.1 Development Platform

The development platform was Windows XP Professional SP2. The code of the prototype was developed in "Oracle JDeveloper Version 10.1.3.1", which is a free integrated development environment, [45] using "Java Platform, Standard Edition Version 5.0" [46]. During the code development, both "embedded OC4J[4] in Oracle JDeveloper" [47] and "Apache Tomcat Version 5.5" [48] were used as Application Server.

As mentioned in chapter 4, "Jena APIs Version 2.5.2" [38] was used as Semantic Web Framework for Java. Both "Oracle Database 10g Express Edition" [49] and "MySQL Version 5.0" [50] databases were used during the code development in order to store and retrieve Semantic data.

"Graphviz Version 2.12" [43] was used as visual graph generator for Semantic data and "Applet version of IsaViz" [51] was used as graph viewer.

All above mentioned tools and softwares are free and available to download from the corresponding web sites.

---

[4] OC4J (Oracle Containers for J2EE) is the core J2EE runtime component of Oracle Application Server. See reference [47].

5.2 System Configuration

This section describes the provided method for administrators to configure UltimateOMS. The administrators define end users and also modify system configuration file.

UltimateOMS enables administrators to modify system configuration by providing a configuration file. The configuration file name is *systemConfig.xml*, which is placed in web application root directory beside the JSP files. The configuration file is a XML file including some parameters related to the graph and available databases as shown in figure 5.1.

```
<?xml version="1.0" encoding="windows-1252" ?>
<SystemConfig>
    <DatabaseList>
        <database>
            <description>MySQL</description>
            <id>MySQL</id>
            <driver>com.mysql.jdbc.Driver</driver>
            <url>jdbc:mysql://localhost/test</url>
        </database>
        .
        .
        .
        <database>
            <description>Oracle</description>
            <id>Oracle</id>
            <driver>oracle.jdbc.driver.OracleDriver</driver>
            <url>jdbc:oracle:thin:@orcl</url>
        </database>
    </DatabaseList>
    <GraphParameters>
        <GRAPH_PANEL_WIDTH>1000</GRAPH_PANEL_WIDTH>
        <GRAPH_PANEL_HEIGHT>600</GRAPH_PANEL_HEIGHT>
        <SERVLET_TMP_DIR>D:/SemanticWeb/UltimateOMS/public_html/temp</SERVLET_TMP_DIR>
        <SERVLET_TMP_DIR_ALIAS>temp/</SERVLET_TMP_DIR_ALIAS>
        <GRAPH_VIZ_ROOT>D:/Program Files/ATT/Graphviz</GRAPH_VIZ_ROOT>
        <GRAPH_VIZ_PATH>D:/Program Files/ATT/Graphviz/bin/dot.exe</GRAPH_VIZ_PATH>
        <GRAPH_VIZ_FONT_DIR>D:/Program Files/ATT/Graphviz</GRAPH_VIZ_FONT_DIR>
    </GraphParameters>
</SystemConfig>
```

Figure 5.1: Part of the System Configuration file of UltimateOMS

5.2.1 Database

As mentioned in Section 4.3.3, a wide range of databases is supported by Jena Semantic Web framework. Therefore, depending on the end users choice, the administrator would be able to install and configure necessary databases.

Administrators can easily add new databases to UltimateOMS and remove them from UltimateOMS by modifying database parameters in system configuration file as shown in the *DatabaseList* section in figure 5.1.

The database parameters in the system configuration file are:

- **description**: a brief description about the database
- **id**: the name of the database which is used in Jena
- **driver**: proper driver of the database
- **url**: URL of the database

UltimateOMS will use the above parameters in order to connect to the database. The administrator will have a choice to install required databases either on the same server as UltimateOMS or on separate servers.

5.2.2 Graph

UltimateOMS will interface with Graphviz in order to generate visual graph of the Semantic data. Consequently, Graphviz has to be installed on the same server as UltimateOMS resides on. The administrator can easily provide Graphviz installed path directory to UltimateOMS by modifying identified parameters in the system configuration file. Since, both Graphviz and UltimateOMS use a shared area in the server in order to keep temporary files for communicating with each other,

specific parameters are also provided in the system configuration file and the administrator can modify them. In our case, the shared area would be a specific directory, which is accessible by Graphviz and UltimateOMS.

There are also some parameters in system configuration file for setting the width and height of the panel in which graph will be shown.

All mentioned graph parameters are in the *GraphParameters* section as shown in figure 5.1.

The graph parameters in system configuration file are:

- ***GRAPH_PANEL_WIDTH***: The width of the panel in which the graphs will be shown (in pixel).
- ***GRAPH_PANEL_HEIGHT***: The height of the panel in which the graphs will be shown (in pixel).
- ***SERVLET_TMP_DIR***: The temporary directory path, which is used to save the graph temporary files.
- ***SERVLET_TMP_DIR_ALIAS***: *Alias* which is defined in application server and it is a URL mapping in order to access *SERVLET_TMP_DIR* contents by application server.
- ***GRAPH_VIZ_ROOT***: Path of Graphviz root directory.
- ***GRAPH_VIZ_PATH***: Path of the *dot.exe* directory in Graphviz.
- ***GRAPH_VIZ_FONT_DIR***: Path of the font directory in Graphviz.

5.2.3 Users

The administrator defines the end users of UltimateOMS by creating users in databases. By using database administration's tool and logging in database with admin role, the administrator would be able to create database users. Created database users are application's end users and they can access UltimateOMS by logging in to the application through the login page. Therefore, end users of UltimateOMS directly depend on defined database users and by removing a specific database user, that user is no longer able to access UltimateOMS.

For authenticating a user, UltimateOMS tries to establish a database connection using provided username, password, and type of the database. If the user is a valid user in the specified database, then user will have access to UltimateOMS. Otherwise, the authentication process will fail and user will be informed by proper messages.

5.3 JSP and Java Servlets

All the features and user requirements, which are specified in Use Case Models in Appendix B, are implemented using JSP and Java Servlets technologies. All user interfaces are designed by JSP technology using JSP Standard Tags, HTML Tags, "Jakarta Input Tag Library" [52], and CSS (Cascade Style Sheets).

UltimateOMS business logics are implemented using Java Servlets and Java Beans except for some cases in which the business logic is simple where it is implemented within JSP pages.

Jena APIs are used when it is necessary to deal with Semantic data.

UltimateOMS implementation consists of 65 JSP pages, 25 Java Classes including Java Servlets and Java Beans, one Cascade Style Sheets, and

one system configuration file. All the Java Classes reside in Java Package named *se.kth.ict.ultimateoms*.

5.3.1 Authentication

Authentication is done using username and password provided by administrators. Using Login Page end users enter their usernames and passwords and identify the database, to which they want to connect. UltimateOMS authenticates end users by establishing a database connection using received information. If the database connection is successfully established then the user is a valid user in the specified database and the user will have authorization to access UltimateOMS.

If the authorization was successful, UltimateOMS will create a new Session Bean for connected user in order to store session data and status. The Session Management is presented in section 5.3.2.

Using SSL (Secure Socket Layer) protocol it is quite possible to make communication between end users and UltimateOMS more secure. Since the security was not the crucial part of UltimateOMS, this part it is not implemented.

5.3.2 Session Management

All the information and status related to the connected user are kept in a Session Bean. By storing session data in the Session Bean, UltimateOMS can keep track of the connected users' requests. The Session Bean is a Java Class and named *SessionBean.java* in UltimateOMS package. It will be created for each connected end user and is used in almost all JSP pages using the following JSP Tag:

```
<jsp:useBean id="session Bean"
             class="se.kth.ict.ultimateoms.SessionBean"
             scope="session"/>
```

As shown in the JSP Tag, the scope of the Session Bean is *Session*, which means that the created Session Bean is valid as long as the session between client web browser and UltimateOMS Application Server is not expired. If end users do not have interaction with UltimateOMS for specific amount of time, which is adjustable in the Application Server, the Session Bean will be expired for security reasons and the end users will be forwarded back to the Login Page.

The Session Bean consists of some Java variables and *Setter/Getter* methods in order to set the variables and get the variables respectively. The Session Bean keeps some data such as *username, password, databaseName*, and all other requires data, which are necessary for UltimateOMS. A small part of the Session Bean is shown in figure 5.2.

```
.
.
.
private String userName;
private String password;
private String databaseName;
.
.
.
public void setUserName(String value) {
    userName = value;
}
public String getUserName() {
    return userName;
}

public void setPassword(String value) {
    password = value;
}
public String getPassword() {
    return password;
}

public void setDatabaseName(String value) {
    databaseName = value;
}
public String getDatabaseName() {
    return databaseName;
}
.
.
.
```

Figure 5.2: Part of the Session Bean used in UltimateOMS

The Session Bean also keeps the parameters of the system configuration file, which are specified by the administrator. By keeping the system configuration parameters in the Session Bean, UltimateOMS accesses the file system once for each user. By using this method, it is not necessary to read the system configuration file for each user request. This method optimizes the performance of the application.

Setting system configuration parameters in the Session Bean will be done once for each user during the authentication process by using provided methods in *Util Java Class*, which is part of UltimateOMS package. For instance, the *setGraphSettings* method in *Util Class* reads

the system configuration file, *systemConfig.xml*, in order to parse the XML file and set the proper variables in the Session Bean. Therefore, whenever it is necessary to have the graph parameters in UltimateOMS, the value of the related variable in the Session Bean will be used.

5.3.3 Validation and Error Handling

The user data are validated both in the client side and in the server side.

In client side, the validation is done using JavaScripts. JavaScripts will check user data in some cases and alert users if validation error is found. For instance, if the value of a specific field in HTML form cannot be empty, the proper JavaScript makes sure that specified field is not empty before sending it to UltimateOMS.

The server side validations are done in the Java Servlets and Java Beans depending on the related business logic.

UltimateOMS uses JSP framework error handler in order to handle the raised exceptions and errors in the application. In our case a specific JSP error handler page, *errorPage.jsp*, is responsible for catching the raised exceptions in the application and showing the proper message to the end users. Each JSP page identifies its error handler using the *errorPage* attribute in the following JSP Tag:

```
<%@ page language="java" contentType="text/html;
         charset=ISO-8859-1" errorPage="/errorPage.jsp"%>
```

There is also an Exception Java Class defined in UltimateOMS in order to throw a new exception in application explicitly. Whenever it is necessary, the business logics in UltimateOMS can throw a new exception with a proper message by using *UltimateOMSException* Java Class in order to show a specific error to the end user. Those explicit exceptions will be caught and shown to the end user by *errorPage.jsp*.

5.4 Graph Visualization

This section describes the method used in UltimateOMS for generating and displaying visual graph of Semantic data.

5.4.1 Graph Generator

As mentioned earlier, UltimateOMS will interface with Graphviz in order to generate visual graphs of Semantic data. The method used in UltimateOMS is the same as the method used in W3C for generating visual graphs, which is demonstrated in a Java Servlet example, *ARPServlet.java* that implements W3C RDF validation service [53]. In addition to the RDF validation, this Java Servlet parses the RDF data using Jena APIs and generate visual graph using Graphviz.

Some parts of the *ARPServlet.java* were used as a base code, some modifications were made, and a new Java class, *Graph.java*, was made to fulfill the graph requirements in UltimateOMS. In *Graph* class, the identified model that contains Semantic data is parsed using Jena APIs in order to extract all resources such as subjects, predicates, and objects. During the parsing process, a graph description file will be generated as an input file for Graphviz. The generated graph description file will be stored in a specific directory, which is accessible by both UltimateOMS and Graphviz. As mentioned in the Section 5.2.2, the directory path can be set and identified using graph parameters in the system configuration file.

The suitable layout program of Graphviz, which is "dot" in our case, will be executed using Java Runtime Class in order to generate visual graph. By executing a command, Graphviz will take the generated graph description file as an input and generate visual graph as SVG (Scalable Vector Graphics) format. The generated SVG file will be stored in the shared directory.

If the end user intention was seeing the generated graph in a web browser, Graph class will send the generated SVG file to the user through a Java Applet application, which is described in Section 5.4.2.

There is also the option to export the generated graph in different formats. In order to generate and export user's desired graph format, a Java Servlet class, *GraphExportServlet.java*, will parse the identified model that contains the Semantic data in the way described earlier. In this case instead of generating SVG format, *Graph Export Servlet* class will execute the layout program of Graphviz, "dot", with proper parameters in order to generate the desired graph format. The generated visual graph will be sent to the user.

The visual graph of the Semantic data can be generated and exported in the following formats:

- PNG (Portable Network Graphics)
- SVG (Scalable Vector Graphics)
- GIF (Graphics Interchange Format)
- PostScript

5.4.2 Graph Viewer

As mentioned in the Section 5.4.1 when the end user intention is seeing the generated visual graph in a web browser, SVG format of the generated graph will be sent to the user through a Java Applet application (see figure 5.3). Therefore, it is necessary to provide a user interface for the end users allowing them to have smooth zooming and navigation in the graph.

Figure 5.3: UltimateOMS visualization graph

IsaViz [51] is a visual environment that provides these capabilities. Since the end users need a visual environment in their web browsers, Applet version of the IsaViz is used in UltimateOMS. As a result, after generating SVG format of visual graph, using the method described in the Section 5.4.1, dynamic Applet tags will be generated in HTML and sent to the end users. In the client side, the web browser will load the Applet application, which is identified in the received Applet tags, and displays the generated visual graph.

Sample of the dynamically generated Applet tags is shown in figure 5.4.

```
<applet code="org.w3c.IsaViz.applet.IsvBrowser.class"
        archive="isaviz.jar,zvtm.jar,xercesImpl.jar,xmlParserAPIs.jar"
        codebase="http://localhost:8988/UltimateOMS/lib"
        width="1000" height="600">

    <param name="type" value="application/x-java-applet;version=1.4"/>
    <param name="scriptable" value="false"/>
    <param name="width" value="1000"/>
    <param name="height" value="600"/>
    <param name="svgFile" value="http://localhost:8988/UltimateOMS/temp/graph_1.svg"/>

</applet>
```

Figure 5.4: Sample of the dynamically generated Applet tags

As shown in the figure 5.4, the class of the Applet is indicated in *code* attribute. *archive* attribute contains all Java Archives that are necessary for running the Applet. The generated SVG format of the visual graph is identified as a parameter, which is the input for the Applet. In addition, the width and height of the panel in which graph will be shown, are indicated as input parameters for the Applet. As mentioned in the Section 5.2.2, the administrator identifies these parameters by using graph parameters in the system configuration file.

5.5 Management and Manipulation

UltimateOMS provides the necessary features for management and manipulation of Semantic data using Jena APIs. Those features are accessible from several menu items. In order to make it a user-friendly interface, those menu items are categorized under five menu groups: Models, Triples, Classes, Properties, and Individual, as shown in figure 5.5.

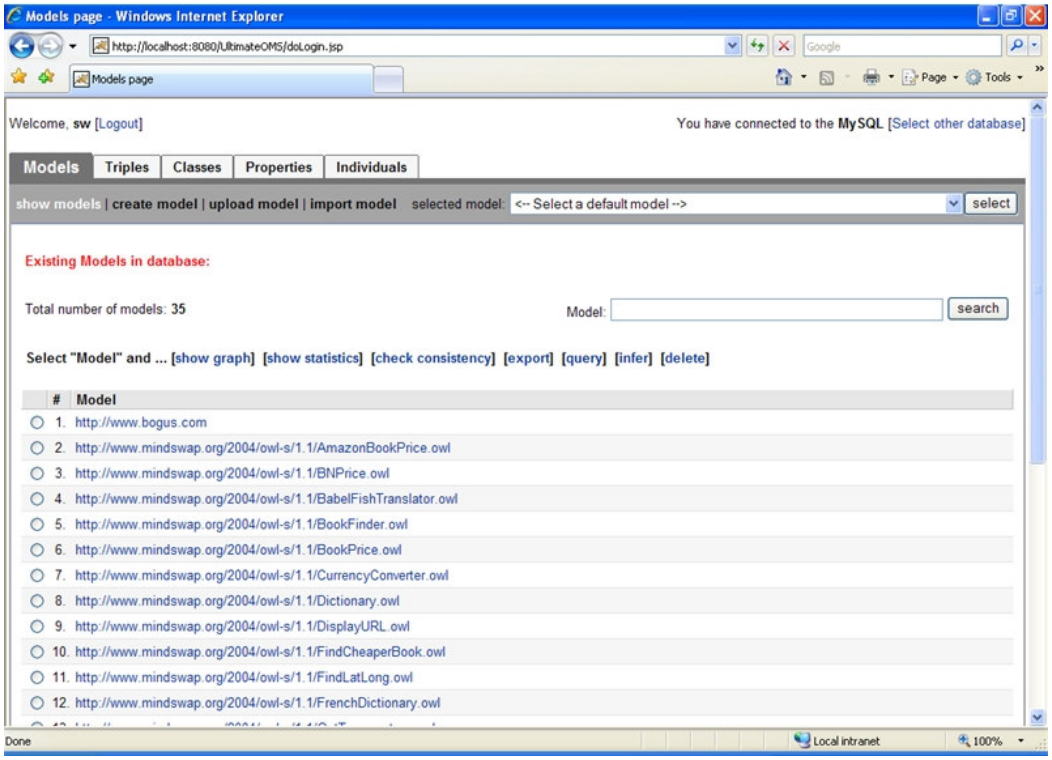

Figure 5.5: Categorized menu groups in UltimateOMS for management and manipulation of Semantic data

5.5.1 Models

In UltimateOMS RDF, RDF Schema, and OWL files are called Models, which are stored in databases. As shown in figure 5.5 the menu group Models contains features for management of models such as creating new model in the database, uploading and importing an existing RDF, RDF Schema, or OWL files as a new model.

Furthermore, additional features are provided inside the page to facilitate managing and manipulating models. These additional features include: searching for specific models in database, selecting a default model to manipulate, showing visualization graph of a model, showing statistics about a model (including the number of triples, classes, properties, and individuals), checking model consistency, exporting a model (in the formats of RDF/XML, Notation3/N3, and N-Triples), querying model using SPARQL, inferring model with different types of inference provided by Jena, and deleting model from database.

Graph visualization, inference, and querying features are described in sections 5.4, 5.6, and 5.7 respectively.

In order to create a new model in UltimateOMS, the method *createModel, which is in ModelMaker interface of Jena APIs,* is used. *ModelMaker* interface contains some methods for creating new models or opening previously created models. To create a new model in the database, it is necessary to create an instance of *ModelMaker* interface, which directly creates models in database or opens existing models in database.

ModelFactory class in Jena can be used to create our desired *ModelMaker*. The following code display how to create a *ModelMaker* using mentioned methods:

```
Class.forName(databaseDriver);
dbConnection = new DBConnection(databaseURL, userName,
                                password, databaseName);
modelMaker = ModelFactory.createModelRDBMaker(dbConnection);
```

In the above code, a new connection to the desired database is established using constructor *DBConnection*, which is available in *DBConnection* class of Jena APIs. Using the established connection, *createModelRDBMaker* method in *ModelFactory* can create a *ModelMaker*. This way, by creating *ModelMaker* and using proper methods of *ModelMaker*, we can create new models in database or open existing models in database.

UltimateOMS uses *read* method of *Model* interface from Jena APIs in order to import or upload RDF, RDF Schema, or OWL files as new models in databases. It also uses *write* method of *Model* interface for exporting the existing models in databases in form of a file in a selected format. *Model* Interface is defined in package *com.hp.hpl.jena.rdf.model* and provides some methods for dealing with models such as *read* (for reading RDF, RDF Schema, or OWL files) and *write* (for writing all statements in models as a file). UltimateOMS supports different file

formats like RDF/XML, Notation3/N3, and N-Triples for importing or uploading a file as a new model and also for exporting an existing model as a file.

5.5.2 Triples

Each model in UltimateOMS consists of a set of statements. Because each statement has three parts (subject, predicate, and object), it is simply called *triple* in UltimateOMS. As shown in figure 5.6, the menu group, *Triples*, contains some features for managing and manipulating triples, which include creating a new triple in a model, editing an existing triple in a model, deleting a triple from a model, browsing triples, and searching specific triples.

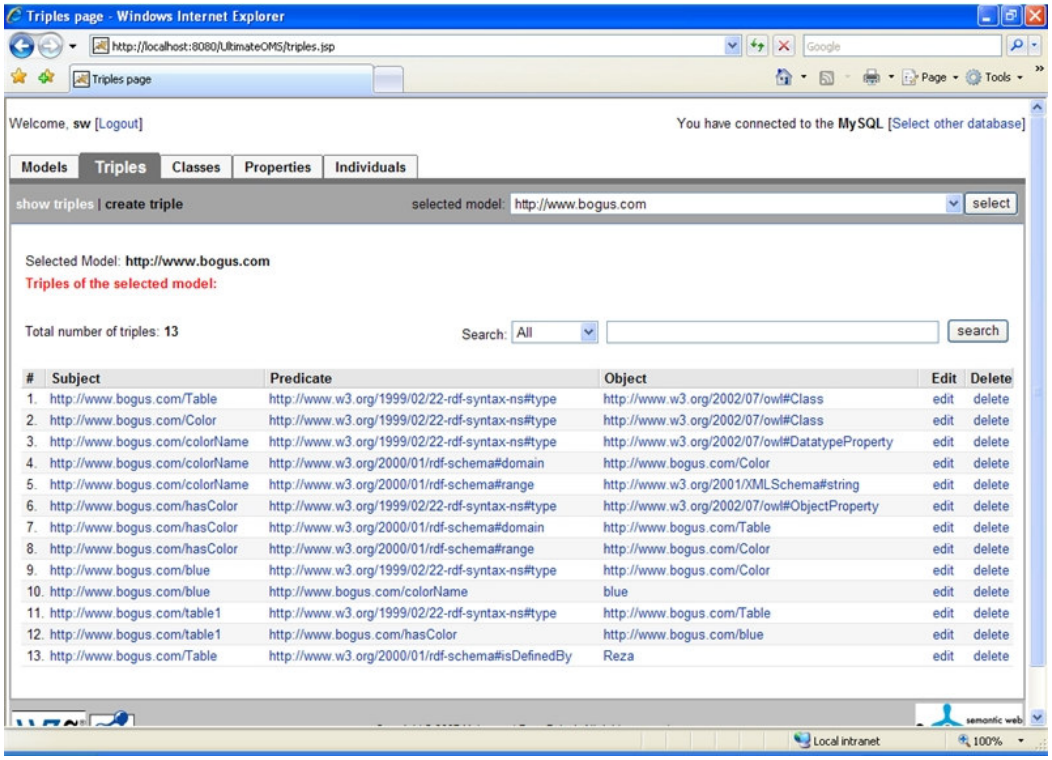

Figure 5.6: Triples menu group in UltimateOMS for management and manipulation of triples

For creating a new triple in a model, UltimateOMS uses provided methods of the *Model* interface from Jena APIs. In the triple creation form, users provide necessary information about the triple including

subject, predicate, and object either by typing or by selecting from a pop-up window. The pop-up window will be created by *listResources.jsp* and contains the list of existing resources in the model as well as vocabulary terms of RDF, RDF Schema, and OWL.

RDF, RDF Schema, and OWL vocabularies as well as datatype definitions for the XML Schema datatypes are defined in a class called *Vocabulary* in UltimateOMS. When the vocabulary terms or XML Schema datatypes are needed, UltimateOMS uses the appropriate defined methods such as *GetRDFVocabulary*, *GetRDFSVocabulary*, *GetOWLVocabulary*, and *GetXSDDatatypes* in *Vocabulary* class.

When UltimateOMS receives the provided information about the subject, predicate, and object, a new triple will be generated in the model using Jena APIs. As an example, the following code shows how to create a triple using provided methods of the *Model* interface:

```
Resource resource = model.createResource(subject);
Property property = model.createProperty(predicate);
Literal literal = model.createTypedLiteral(object, dataType);
Statement stmt = model.createStatement(resource, property,
                                       literal);
model.add(stmt);
```

In the above code, a new resource for *subject* of the triple will be created in the model using *createResource* method of *Model* interface. Using *createProperty* method, a new property will be created in the model as *predicate* of the triple. Since we wanted to create a typed literal as *object* of the triple, *createTypedLiteral* method is used in above code. By having a new resource for *subject*, new property for *predicate*, and new literal for *object* and using *createStatement* method, a new triple will be created and added to the model using *add* method of UltimateOMS.

Although there is no API for editing the triples in Jena, but editing a triple is achievable by simply removing the old triple and creating the new one with modified values for subject, predicate, and object.

5.5.3 Classes

Classes are used to build basic concepts in ontology. As shown in figure 5.7, the menu group *Classes,* includes some features for managing and manipulating of *classes*. It includes creating a new *class* in a *model*, editing an existing *class* in a *model*, deleting a *class* from a *model*, creating a new *instance* for a *class*, showing detail information about a *class*, showing related *triples*, and searching for specific *classes*.

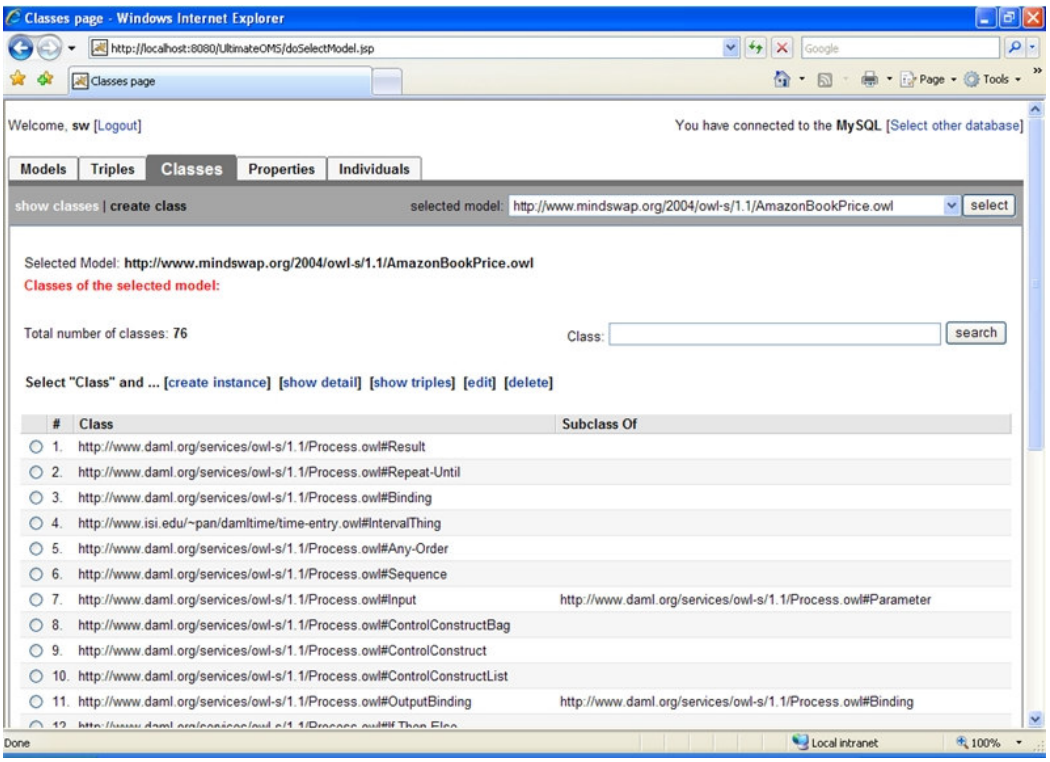

Figure 5.7: Classes menu group in UltimateOMS for management and manipulation of classes

To create a new *class* in a *model*, UltimateOMS uses the methods of *OntModel* interface of Jena APIs. In the *class creation form*, users provide necessary information about the *class*, including *class's*

properties constraints (minimum and maximum cardinality), *class's* axioms (including subClassOf, superClassOf, equivalent, and disjoint), *class's* descriptions (including intersection, union, and complement), and *class's* meta-information (like labels, comments, and annotation properties).

UltimateOMS lists all available *properties* of a *class* in *class creation form* and users can identify the cardinality constraints of them. In addition, values for *class's* axioms and *class's* descriptions should be selected from a new pop-up window. The pop-up window will be created by *listClasses.jsp* and contains a list of all existing *classes* in a *model*.

When UltimateOMS receives provided information about the *class*, the new *class* will be generated in the *model* using the Jena APIs. As an example, the following code shows how a *class* is created using methods of *OntModel* interface of Jena APIs:

```
Model model = modelMaker.openModel(selectedModel);
OntModelSpec spec = new OntModelSpec(OntModelSpec.OWL_MEM);
spec.setBaseModelMaker(modelMaker);
OntModel ontModel = ModelFactory.createOntologyModel(spec,
                                                      model);
newClass = ontModel.createClass(className);
```

In the above code, existing *model* in the database will be opened by *modelMaker* in order to create a new *class*. Since we want to create an *ontology class*, a new *ontology model*, *ontModel*, has to be created on top of the model using *createOntologyModel* method of *ModelFactory* class. To create and *ontology model* it is necessary to specify the type of ontology. This can be done using the class, *OntModelSpec,* and identifying the type of ontology as OWL. After having the *ontology model*, *ontModel*, a new *ontology class* will be created and added to the *model* using *createClass* method of *OntModel* interface.

Although there is no API for editing the classes in Jena, editing a *class* is achievable by removing the *class's* old information and adding a new one with modified value.

Similar to the technique used for creating a *class*, UltimateOMS uses methods of *OntModel* interface of Jena APIs in order to create an *instance* of a *class* in a *model*. *Instances* are called *Individuals* in UltimateOMS. In the *instance creation form*, the user provides necessary information about the *instance* including *instance's* properties and *instance's* meta-information (like labels, comments, and annotation properties). All the available properties of a *class* will be properties of the instance. Therefore, UltimateOMS lists those properties in the *instance creation form* and users can identify values for the properties by either typing (when the range of property is literal or typed literal) or by selecting from a list (when the property range is a list of instances for a specific *class*).

The instances, which were created for each *class* in each specific model, are available under *Individuals* menu group in UltimateOMS.

5.5.4 Properties

As shown in figure 5.8, the menu group called *Properties* in UltimateOMS includes some features for management and manipulation of the *properties* such as creating a new *property* (either object property or datatype property) in a *model*, editing an existing *property* in a *model*, deleting a *property* from a *model*, showing detail information about a *property*, showing related *triples*, and searching for specific *properties*.

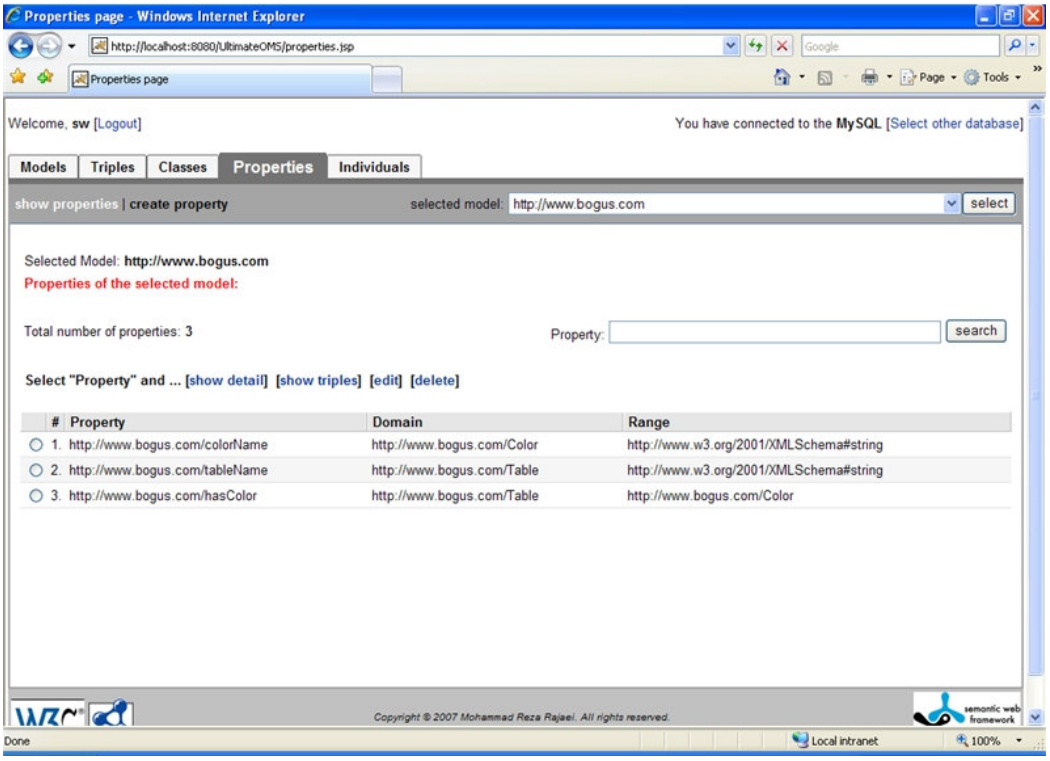

Figure 5.8: Properties menu group in UltimateOMS for management and manipulation of properties

For creating a new *property* in a *model*, UltimateOMS uses some methods of *OntModel* interface of Jena APIs. In the *property creation form*, users provide necessary information regarding a *property*, including *property's* domains and ranges, *property's* characteristics (including functional, inverse functional, transitive, and symmetric), *property's* axioms (including subPropertyOf, superPropertyOf, equivalent, and inverse), and *property's* meta-information (like labels, comments, and annotation properties).

Properties can be defined as a *datatype property* or *object property*. *Values* for *property* domains in both cases have to be selected from a pop-up window, which is created by *listClasses.jsp* in UltimateOMS and contains the list of all existing *classes* in a *model*. If the *property* is an *object property*, then the range of *property* is a *class* and like domain, its values have to be selected from a new pop-up window.

In case of defining *datatype property*, range values can be selected from a list, which contains XML Schema datatypes. As mentioned earlier in section 5.5.2, UltimateOMS uses method *GetXSDDatatypes* of class *Vocabulary* in order to create the list.

Values for *property's* axioms have to be selected from another pop-up window, which is created by *listProperties.jsp* in UltimateOMS and contains list of all existing properties in a *model*.

When UltimateOMS receives provided information about the *property*, the new *property* will be generated in the *model* using the Jena APIs. As an example, the following code shows how to create a datatype property using methods of the *OntModel* interface from Jena APIs:

```
Model model = modelMaker.openModel(selectedModel);
OntModelSpec spec = new OntModelSpec(OntModelSpec.OWL_MEM);
spec.setBaseModelMaker(modelMaker);
OntModel ontModel = ModelFactory.createOntologyModel(spec,
                                                      model);
newProperty = model.createDatatypeProperty(property);
```

In the above code, the existing *model* in the database will be opened by *modelMaker* in order to create a new *property*. Since we want to create an *ontology property*, a new *ontology model*, *ontModel*, has to be created in the way described in section 5.5.3. After having the *ontology model* (*ontModel*), a new *ontology datatype property* will be created and added to the *model* using *createDatatypeProperty* method of *OntModel* interface.

There is no API for editing *properties* in Jena, but editing a *property* is achievable by removing the *property's* old information and adding the new ones with modified values.

5.5.5 Individuals

As shown in figure 5.9, *Individuals* menu group in UltimateOMS includes some features for management and manipulation of the *individuals* such as editing an existing *individual* in a *model*, deleting an *individual* from a *model*, showing detail information about an *individual*, showing related *triples*, and searching for specific *individuals*.

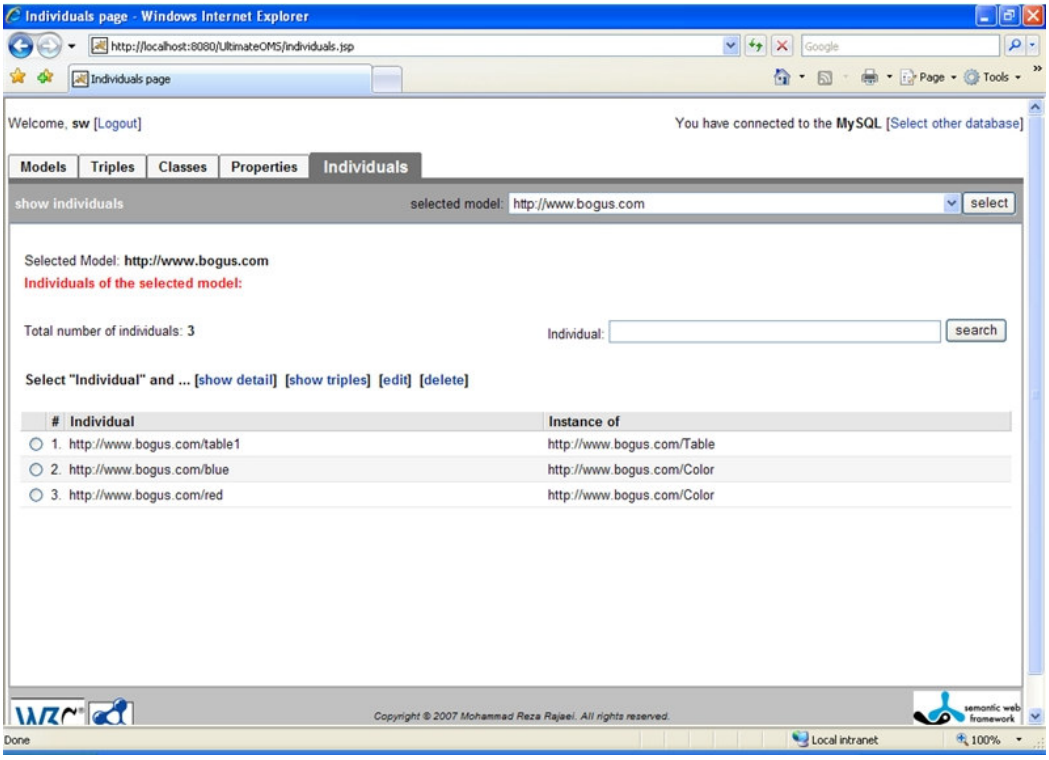

Figure 5.9: Individuals menu group in UltimateOMS for management and manipulation of individuals

As mentioned in section 5.5.3, using the features in *Classes* Menu group of UltimateOMS we can create an *instance* (*individual*) of a *class* in a *model*. The created *individuals* can be edited using *individual creation form*.

Although there is no API for editing the *individuals* in Jena, editing an *individual* is achievable by removing the *individual*'s old information and adding the new ones with modified values.

5.6 Inference

UltimateOMS provides inference facilities using different supported built-in types of "Reasoner" and "Generic Rule Reasoner" of Jena. As shown in figure 5.10, various types of reasoners are available in Jena.

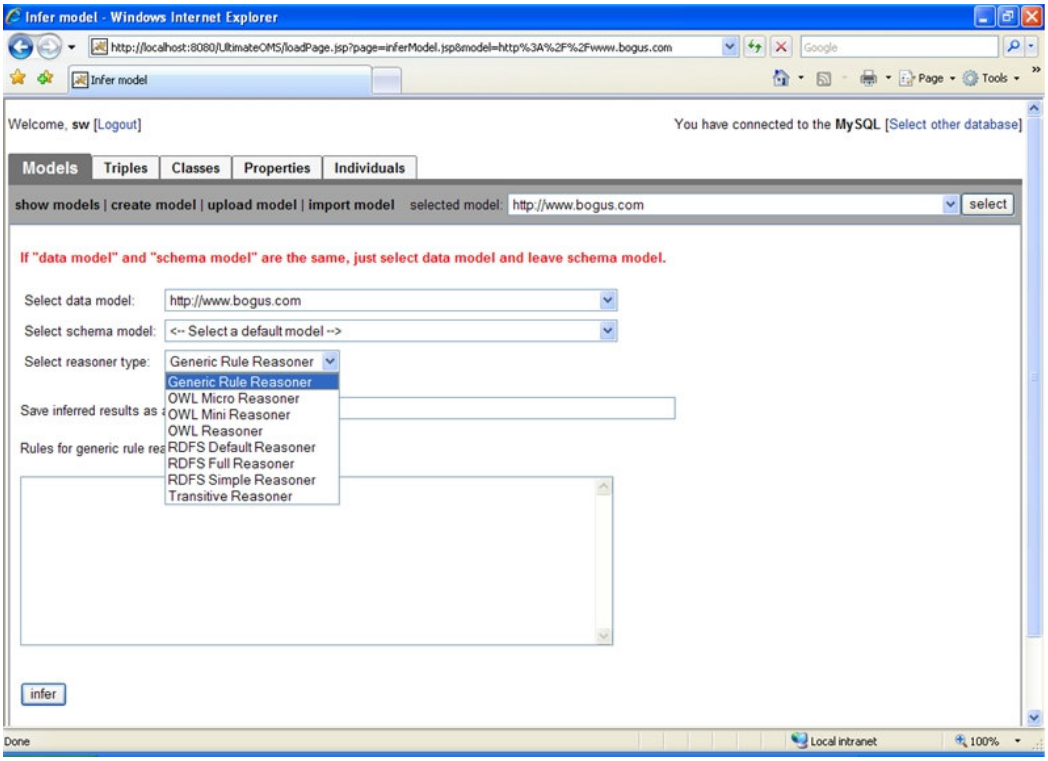

Figure 5.10: Inference facilities in UltimateOMS

Users can infer an existing *model* in the database by using an appropriate reasoner and storing inferred results as a new *model* in the database for further usage. Currently Jena supports the following reasoners:

- Generic Rule Reasoner
- OWL, OWL Mini, OWL Micro Reasoners
- RDFS Rule Reasoner
- Transitive Reasoner

Generic rule reasoner supports user-defined rules. *RDFS rule reasoner* implements subset of RDFS entailments. *OWL, OWL mini, and OWL micro reasoners* support a set of useful but incomplete implementation of OWL Lite. *Transitive reasoner* supports only transitive and symmetric properties of *rdfs:subPropertyOf* and *rdfs:subClassOf*.

Although Jena does not provide complete and sophisticated reasoners, it allows ranges of inference and reasoner engines to be plugged into it by providing DIG interface. The Jena DIG interface makes it possible to use external reasoners such as Racer, Pellet, and FaCT, which support DIG standard.

UltimateOMS only supports the built-in supported reasoners in Jena and adding external reasoners to system using DIG interface is suggested as a future work of this project.

5.7 Querying

Jena supports SPARQL as a standard RDF query language. Query APIs in Jena are mainly located in *com.hp.hpl.jena.query* package. UltimateOMS uses these APIs in order to provide query facilities for users. Users can see the query results online using valid SPARQL query or they can export query results into local file system in different formats.

Figure 5.11 shows *query* user interface of UltimateOMS.

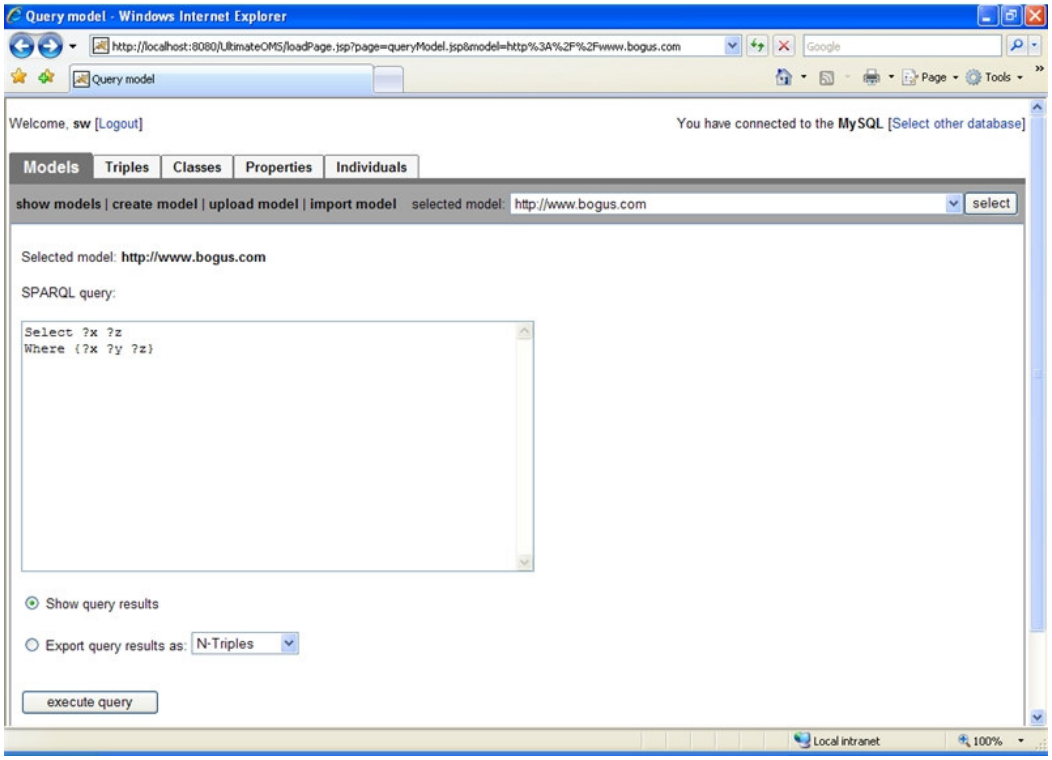

Figure 5.11: Query user interface of UltimateOMS

The SPARQL query results can be exported in the following formats:

- N-Triples
- Notation3/N3
- RDF/XML
- Text File
- XML File

5.8 User Interface

This section presents the designed user interface of UltimateOMS. All user interfaces are designed using JSP technology and JSP Standard tags, HTML tags, "Jakarta Input Tag Library", and CSS (Cascade Style Sheets).

CSS is a Stylesheet Language used for describing HTML, XHTML, and XML documents and allows a flexible design options. A default CSS is used in all user interfaces of UltimateOMS and it can easily be changed by administrators in order to make different Stylesheets.

A part of the used CSS is shown in figure 5.12.

```
/**
    Common elements
**/
body {
    margin:0px 3px;
    font:83% arial,helvetica,clean,sans-serif;
    color:black;
    background-color:#ffffff;
}
/**
    Menu Tabs
**/
div.menu {
    margin-top:.5em;
    height:2.1lem;
    border-style:solid;
    border-width:0 0 1px 0;
    background-color:#737373;
    border-bottom-color:#737373;
}
.
.
.
```

Figure 5.12: A part of the used Stylesheet in UltimateOMS

The designed layout for all user interfaces consists of four different sections including *Header*, *Menu*, *Content*, and *Footer*. As illustrated in figure 5.13, *Header* section contains information regarding connected user and database.

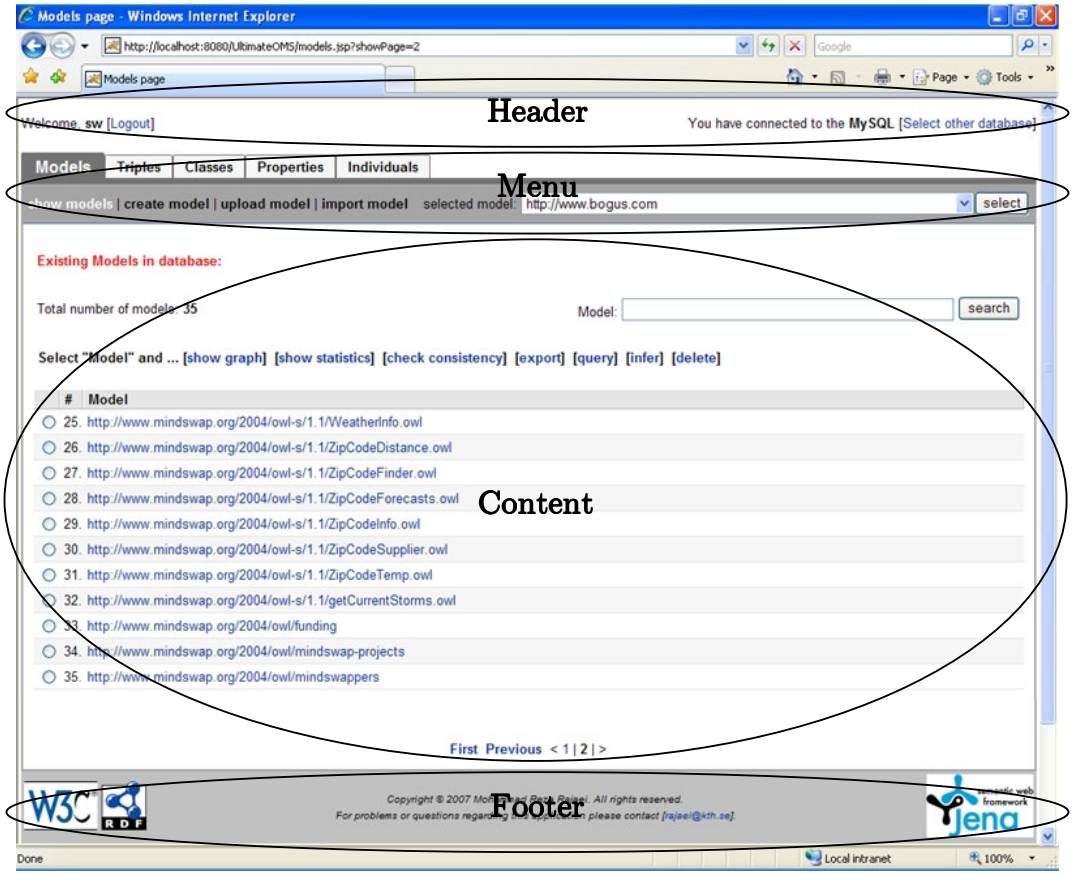

Figure 5.13: Designed layout for UltimateOMS user interfaces

Menu section contains menus and submenus items, which are provided to the end users in order to use UltimateOMS. *Content* section displays the dynamic contents, depends on the end user's requests. *Footer* section contains copyright information and some logos.

5.9 Flow Dynamics

As an example, this section examines the use case *Login* and describes the performed steps by UltimateOMS in order to accomplish authorization process. This use case is described in detail in Appendix B.

In order to access, UltimateOMS end users have to login through the *Login* page as shown in figure 5.14.

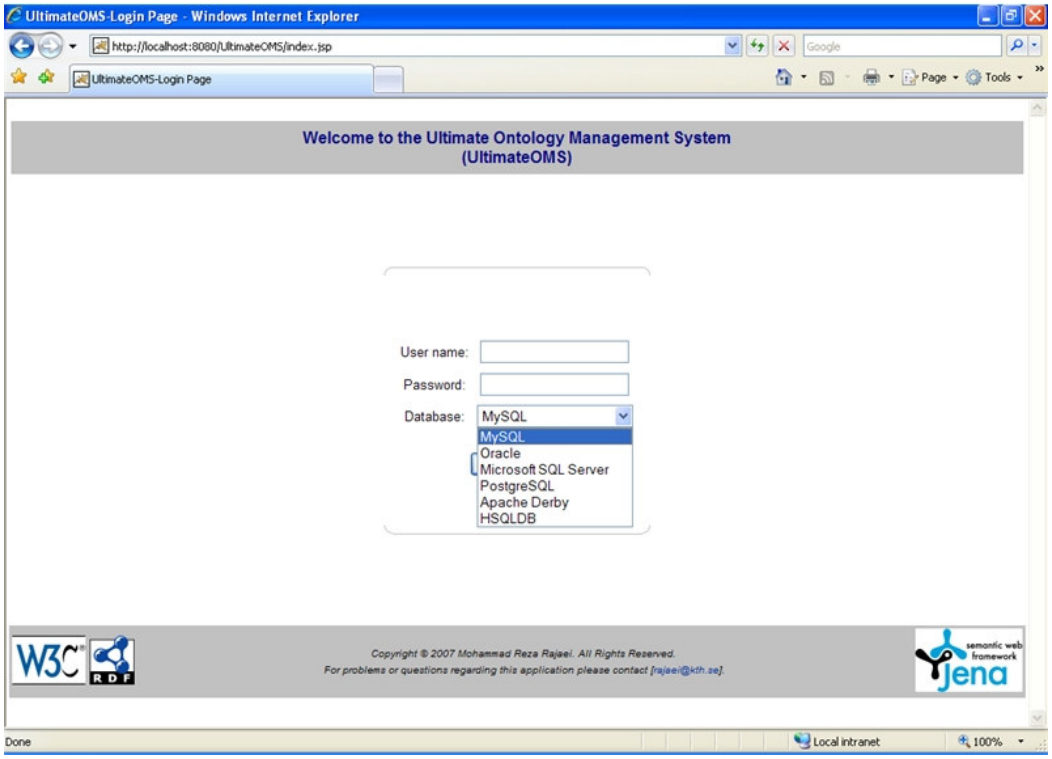

Figure 5.14: UltimateOMS login page

Login Page, *index.jsp*, is a JSP file and when it is being loaded, following actions take place. An instance of the *se.kth.ict.ultimateoms.SessionBean*, which is *sessionBean*, is created as a Session Bean in order to keep all necessary information regarding user requests and application states. The next step is loading available databases from the system configuration file, *systemConfig.xml*, in order to display to the users. This process is done using method *getDatabaseList*, member of *Util* class, by accessing the system configuration file and parsing the databases parameters. Returned list of the available databases are stored in the Session Bean in order to be used in further steps. At this point, all the necessary HTML codes and JavaScripts are generated and sent as a *Login* Page to the user through HTTP Response.

The end user can select one of the available databases and enter his/her username and password in the proper fields. Since username and password cannot be empty when pressing *connect* button, provided JavaScript will check the value of those fields before sending them to the server. If JavaScript validation was successful, the entered field will be sent to the server with HTTP Request.

The Application Server will forward received HTTP Request to *doLogin.jsp*, which is responsible for authorization process in UltimateOMS. By receiving HTTP Request in *doLogin.jsp*, the username, password, and selected database will be stored in Session Bean using following JSP Tags:

```
<jsp:setProperty name="sessionBean" property="databaseName"
                 param="database"/>
<jsp:setProperty name="sessionBean" property="userName"
                 param="userName"/>
<jsp:setProperty name="sessionBean" property="password"
                 param="password"/>
```

The above JSP tags, populate the proper variables in the Session Bean with the values passed through HTTP Request. At this point, *doLogin.jsp* tries to establish a database connection to the selected database. This process is done using *com.hp.hpl.jena.db.DBConnection* Java class that is provided by Jena APIs. *DBConnection* takes database driver, username, password, and database name as input parameters and tries to establish a JDBC connection with specified database.

If the database connection is successfully established and the user is a valid user in the specified database the user will have authorization to access UltimateOMS. The established database connection will be stored in a Session Bean in order to use in further operations. After this step, other variables in the Session Bean including graph parameters that are accessible from the system configuration file, will be initialized and the user will be forwarded to the *Models* page using the JSP tag, `<jsp:forward page="models.jsp"/>`. *Models* page as shown in figure

5.15 is the page that includes all available models in the connected database.

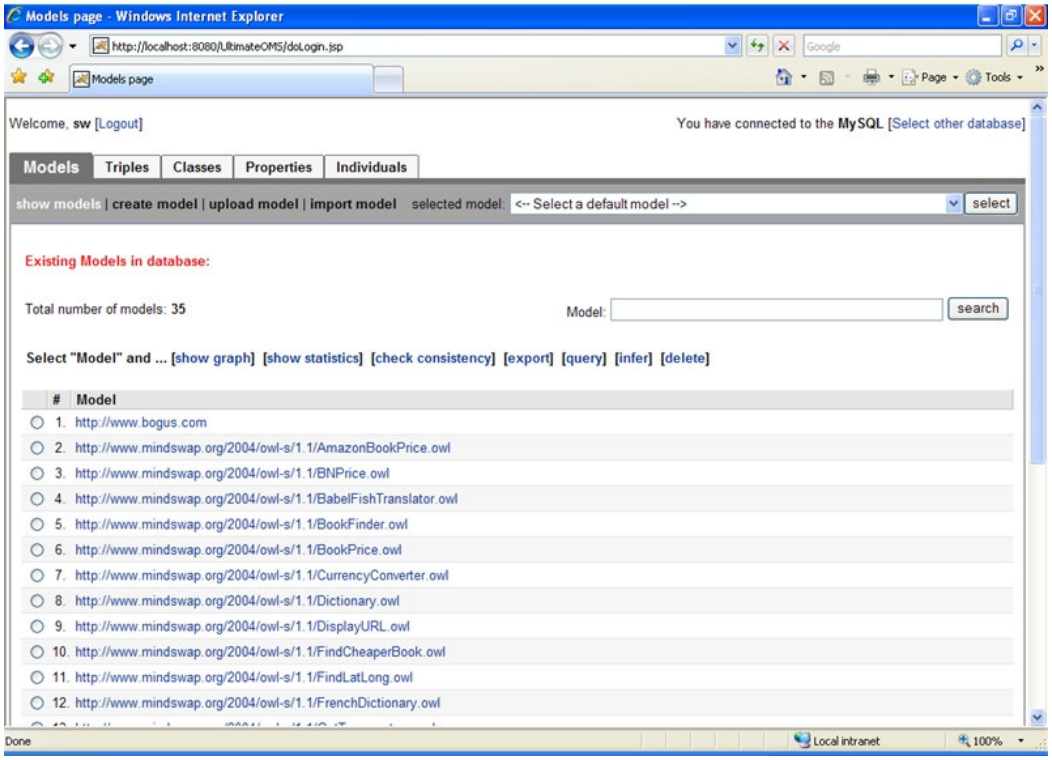

Figure 5.15: UltimateOMS models page

If database connection is not established for any reason, like invalid username and password, *DBConnection* will throw an exception. The thrown exception in *doLogin.jsp* will be handled by JSP error handler page, *errorPage.jsp*, and the proper message with reason of the login failure will be sent to the user.

Chapter 6

Validation

According to the mentioned development platform in section 5.1, the following use cases have been implemented. For further details about the use cases, see appendix B.

- System configuration creation and modification including modifying database parameters, adding new database, removing database, and modifying graph parameters
- User authentication including login and logout
- Model management including showing, creating, importing, uploading, searching, selecting, exporting, showing graph, exporting graph, querying, exporting query results, inferring, showing statistics, checking consistency, and deleting
- Triple management including showing, creating, searching, browsing, selecting, editing, and deleting
- Class management including showing, creating, searching, selecting, showing details, showing triples, creating instance, editing, and deleting
- Property management including showing, creating, searching, selecting, showing details, showing triples, editing, and deleting
- Individual management including showing, searching, selecting, showing details, showing triples, editing, and deleting

The application installation, starting up, and shutting down use cases do not require implementation. Because, for installing UltimateOMS, administrator deploys the application's WAR file into the installed

Application Server using provided tools in Application Server. In addition, by using management tools of Application Server, administrator can start up and shut down UltimateOMS.

Similarly, creating, modifying, and removing database users use case do not require implementation. Because by using database administration's tool and logging in database as an admin role, administrator would be able to create, modify, and remove database's users, which are also UltimateOMS users.

All of the mentioned implemented use cases in the above have been passed the functionality tests without any significant programming error. All implementation and optimization issues are considered during the implementation of the use cases for two different types of Web browsers: Internet Explorer Version 7 and Mozilla Firefox Version 2.0.0.1.

All of the use cases have been tested from different machines using two different mentioned Web browsers. All of them fulfilled their expected behavior and passed the functionality tests with different types of data without any error.

Regarding the reliability and performance of UltimateOMS, in terms of response time, it is limited to the reliability and performance of the Jena Semantic Web Framework as mentioned in section 3.5.

Chapter 7

Conclusions and Future Work

One of the primitive goals of the Web was making the information processable for machines. However, today information on the Web is designed for human interpretation and it is not machine processable. In order to achieve the Web's primitive goal, information on the Web needs to be expressed in a form that machines would be able to understand it instead of simply displaying it and this is exactly the goal of Semantic Web.

Having the vision that Semantic Web is the future of the Web, and taking into account that Semantic Web is still in early steps, it is vital to provide sophisticated tools for users who intend to share machine processable data in the Web by creating, managing, and publishing Semantic data. Nevertheless, almost none of the existing solutions provide a complete tool containing the required features for dealing with Semantic data. This results in a slow and timely process.

In this project, a Web based tool called UltimateOMS was developed that facilitates creating and managing Semantic data in RDF, RDF Schema, and OWL formats. Unlike existing tools, the developed tool brings the necessary functions for creating, manipulating, and managing Ontology components in one place thus making it much easier for users to deal with Semantic data.

UltimateOMS is based on Jena, which is a Java framework for building Semantic Web applications, and is developed using JSP and Java Servlets technology. UltimateOMS gives different options for storing Semantic data in different databases including MySQL, HSQLDB, PostgreSQL, Oracle, Apache Derby, and Microsoft SQL Server. It also provides necessary features for generating visualization graphs of the

Semantic data using Graphviz, the graph visualization software, and displays them using applet version of the IsaViz in web browsers. The generated visualization graph can also be exported in different formats such as PNG, SVG, GIF, and PostScript in order to be used in different tools.

In addition, UltimateOMS has the necessary features in order to create, manipulate, and manage Semantic data as well as features for querying and inferring Semantic data.

A future work of this thesis can be to extend the inferring capability of UltimateOMS by adding external reasoner engines such as Racer, Pellet, and FaCT. Jena currently has its own reasoning engine with limited inferring capabilities. This suggestion can be implemented using DIG interface of Jena that makes it possible to add external reasoners.

Another future work can be using SSL protocol between the end user and UltimateOMS in order to make the communication more secure. In order to increase the security in the cases that the database server is separate from the UltimateOMS server, SSL protocol can also be used between those two servers as well.

References

[1] Tim Berners-Lee, *A roadmap to the Semantic Web*, September 1998, http://www.w3.org/DesignIssues/Semantic.html (last accessed: October 2006)

[2] Nature Publishing Group, *Nature Debates: Scientific publishing on the 'semantic web'*, http://www.nature.com/nature/debates/e-access/Articles/bernerslee.htm (last accessed: October 2006)

[3] Deborah L. McGuinness, Frank van Harmelen eds., *OWL Web Ontology Language Overview*, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-features/ (last accessed: October 2006)

[4] Jeff Heflin ed., *Web Ontology Language (OWL) Use Cases and Requirements*, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/webont-req/ (last accessed: October 2006)

[5] *pOWL - Semantic Web Development Platform*, http://powl.sourceforge.net/ (last accessed: October 2006)

[6] *Sesame*, http://www.openrdf.org/ (last accessed: October 2006)

[7] Stanford, *The Protégé Ontology Editor and Knowledge Acquisition System*, http://protege.stanford.edu/ (last accessed: October 2006)

[8] Wikipedia, the free encyclopedia, *Semantic Web*, http://en.wikipedia.org/wiki/Semantic˙Web (last accessed: October 2006)

[9] James Hendler, Tim Berners-Lee and Eric Miller, "*Integrating Applications on the Semantic Web*", Journal of the Institute of

Electrical Engineers of Japan, Vol 122(10), October, 2002, p. 676-680., http://www.w3.org/2002/07/swint

[10] *The Semantic Web: An Introduction*, http://infomesh.net/2001/swintro/ (last accessed: October 2006)

[11] *Semantic Web Benefits and Demonstration*, http://refapp.semwebcentral.org/tutorial/guided-tour/guidedtourbenefits.html (last accessed: October 2006)

[12] Roberto Garcia Gonzalez, "*A Semantic Web approach to Digital Rights Management*", PhD Thesis, Universitat Pompeu Fabra, Barcelona, November 2005 http://rhizomik.net/~roberto/thesis/Thesis.pdf (last accessed: October 2006)

[13] *The Friend of a Friend (FOAF) project*, http://www.foaf-project.org/ (last accessed: October 2006)

[14] *BigBlogZoo*, http://www.bigblogzoo.com/ (last accessed: October 2006)

[15] *Piggy Bank*, http://simile.mit.edu/wiki/Piggy_Bank (last accessed: October 2006)

[16] World Wide Web Consortium (W3C), *Resource Description Framework (RDF)*, http://www.w3.org/RDF/ (last accessed: October 2006)

[17] Graham Klyne, Jeremy Carroll, eds., *Resource Description Framework (RDF): Concepts and Abstract Syntax*, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/rdf-concepts/ (last accessed: October 2006)

[18] Frank Manola, Eric Miller, eds., *RDF Primer*, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/rdf-primer/ (last accessed: October 2006)

[19] Kevin Wilkinson, Craig Sayers, Harumi Kuno, Dave Reynolds, "*Efficient RDF Storage and Retrieval in Jena2*", First International Workshop on Semantic Web and Databases, Berlin, Germany, 7 September 2003, http://www.hpl.hp.com/techreports/2003/HPL-2003-266.pdf (last accessed: October 2006)

[20] Jeen Broekstra, "*Storage, Querying and Inferencing for Semantic Web Languages*", PhD Thesis, Vrije Universiteit, 4 July 2005, http://wwwis.win.tue.nl/~jbroekst/thesis/thesis-final.zip (last accessed: October 2006)

[21] Dave Beckett, ed., *RDF/XML Syntax Specification (Revised)*, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/rdf-syntax-grammar/ (last accessed: October 2006)

[22] Jan Grant, Dave Beckett, eds., *RDF Test Cases*, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/rdf-testcases/ (last accessed: October 2006)

[23] Dave Beckett, *Turtle - Terse RDF Triple Language*, 10 February 2004, http://www.dajobe.org/2004/01/turtle/ (last accessed: October 2006)

[24] Tim Berners-Lee, *Notation3 (N3) A readable RDF syntax*, 1998, http://www.w3.org/DesignIssues/Notation3 (last accessed: October 2006)

[25] Dan Brickley, R.V. Guha, eds., *RDF Vocabulary Description Language 1.0: RDF Schema*, 10 February 2004, W3C Recommendation, http://www.w3.org/TR/rdf-schema/ (last accessed: October 2006)

[26] World Wide Web Consortium (W3C), *Web Ontology Language (OWL)*, http://www.w3.org/2004/OWL/ (last accessed: October 2006)

[27] Michael K. Smith, Deborah McGuinness, Raphael Volz, Chris Welty eds., *OWL Web Ontology Language Guide*, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-guide/ (last accessed: October 2006)

[28] Wikipedia, the free encyclopedia, *Web Ontology Language*, http://en.wikipedia.org/wiki/Web˙Ontology˙Language (last accessed: October 2006)

[29] *Ontology Inference Layer (OIL)*, http://oil.semanticweb.org/ (last accessed: October 2006)

[30] Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, Lynn Andrea Stein, *DAML+OIL (March 2001) Reference Description*, 18 December 2001, http://www.w3.org/TR/daml+oil-reference (last accessed: October 2006)

[31] Mike Dean, Guus Schreiber eds., Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, Lynn Andrea Stein, *OWL Web Ontology Language Reference*, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-ref/ (last accessed: October 2006)

[32] Peter Haase, Jeen Broekstra, Andreas Eberhart, Raphael Volz, "*A Comparison of RDF Query Languages*", Proceedings of the 3rd International Semantic Web Conference (ISWC2004), 7-11 Nov. 2004, Hiroshima, Japan, http://www.aifb.uni-karlsruhe.de/WBS/pha/rdf-query/rdfquery.pdf (last accessed: October 2006)

[33] Eric Prud'hommeaux, Andy Seaborne, *SPARQL Query Language for RDF*, W3C Working Draft, 4 October 2006,

http://www.w3.org/TR/rdf-sparql-query/ (last accessed: November 2006)

[34] *Rational Unified Process: Vision Artifact*, http://www.ts.mah.se/RUP/RationalUnifiedProcess/process/artifact/ar˙vsion.htm (last accessed: November 2006)

[35] Wikipedia, the free encyclopedia, *IBM Rational Unified Process*, http://en.wikipedia.org/wiki/Rational˙Unified˙Process (last accessed: November 2006)

[36] *IBM Rational Software*, http://www-306.ibm.com/software/rational/ (last accessed: November 2006)

[37] *JavaServer Pages Overview*, http://java.sun.com/products/jsp/overview.html (last accessed: November 2006)

[38] *Jena Semantic Web Framework*, http://jena.sourceforge.net/index.html (last accessed: February 2007)

[39] *Jena Relational Database Backend*, http://jena.sourceforge.net/DB/index.html (last accessed: January 2007)

[40] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, Kevin Wilkinson, "*Jena: Implementing the Semantic Web Recommendations*", Digital Media Systems Laboratory, HP Laboratories Bristol, 24 December 2003, http://www.hpl.hp.com/techreports/2003/HPL-2003-146.pdf (last accessed: November 2006)

[41] *Semantic Web Research at HP Labs*, http://www.hpl.hp.com/semweb/ (last accessed: November 2006)

[42] *Jena2 Database Interface - Database Layout*, http://jena.sourceforge.net/DB/layout.html (last accessed: January 2007)

[43] *Graphviz – Graph Visualization Software*, http://www.graphviz.org/ (last accessed: November 2006)

[44] *Sun Developer Bookshelf-Books: JavaServer Pages*, http://java.sun.com/developer/Books/javaserverpages/Chap12.pdf (last accessed: November 2006)

[45] *Oracle JDeveloper – Official Home Page*, http://www.oracle.com/technology/products/jdev/index.html (last accessed: November 2006)

[46] *Java Platform, Standard Edition (J2SE 5.0)*, http://java.sun.com/j2se/1.5.0/ (last accessed: November 2006)

[47] *Oracle Containers for Java (OC4J)*, http://www.oracle.com/technology/tech/java/oc4j/index.html (last accessed: November 2006)

[48] *Apache Tomcat*, http://tomcat.apache.org/ (last accessed: November 2006)

[49] *Oracle Database 10g Express Edition*, http://www.oracle.com/technology/products/database/xe/index.html (last accessed: November 2006)

[50] *MySQL AB*, http://www.mysql.com/ (last accessed: November 2006)

[51] *IsaViz: A Visual Authoring Tool for RDF*, http://www.w3.org/2001/11/IsaViz/ (last accessed: November 2006)

[52] *The Jakarta Taglibs Project*, http://jakarta.apache.org/taglibs/index.html (last accessed: November 2006)

[53] *CVS log for java/classes/org/w3c/rdf/examples/ARPServlet.java*, http://dev.w3.org/cvsweb/java/classes/org/w3c/rdf/examples/ARPServlet.java (last accessed: November 2006)

Appendices

A - Abbreviations

API	Application Program Interface
CSS	Cascade Style Sheets
DAML	DARPA Agent Markup Language
EAR	Enterprise Archive
EJB	Enterprise Java Bean
FoaF	Friend of a Friend
FOL	First Order Logic
GIF	Graphics Interchange Format
Graphviz	Graph Visualization Software
HTML	HyperText Markup Language
I/O	Input/Output
J2EE	Java 2 Platform, Enterprise Edition
J2SE	Java Platform, Standard Edition
JDBC	Java Database Connectivity
JSP	JavaServer Pages
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MIME	Multi-purpose Internet Mail Extension
OC4J	Oracle Containers for J2EE
OIL	Ontology Inference Layer
OWL	Web Ontology Language
PC	Personal Computer
PHP	PHP Hypertext Preprocessor
PNG	Portable Network Graphics
RDF	Resource Description Framework
RDF Schema	Resource Description Framework Schema
RDFS	Resource Description Framework Schema
RuleML	Rule Markup Language

RUP	Rational Unified Process
SeRQL	Sesame RDF Query Language
SSL	Secure Socket Layer
SVG	Scalable Vector Graphics
SWRL	Semantic Web Rule Language
SWRL FOL	Semantic Web Rule Language First Order Logic
Turtle	Terse RDF Triple Language
UltimateOMS	Ultimate Ontology Management System
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAR	Web Archive
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

B - Use Case Model

B.1 Actors

There are two different types of actors in UltimateOMS application.

- **Administrator**

  The term Administrator applies to a person who is in charge of application. Generally, Administrator creates and modifies application's default configuration file, maintains databases, and also defines and modifies application's users in databases.

- **User**

  The term User applies to anyone, who is defined by Administrator as a valid user in application, using the application in order to create and manage semantic data (ontology components).

B.2 Use Case Diagrams

The following figures show the use case diagrams of UltimateOMS.

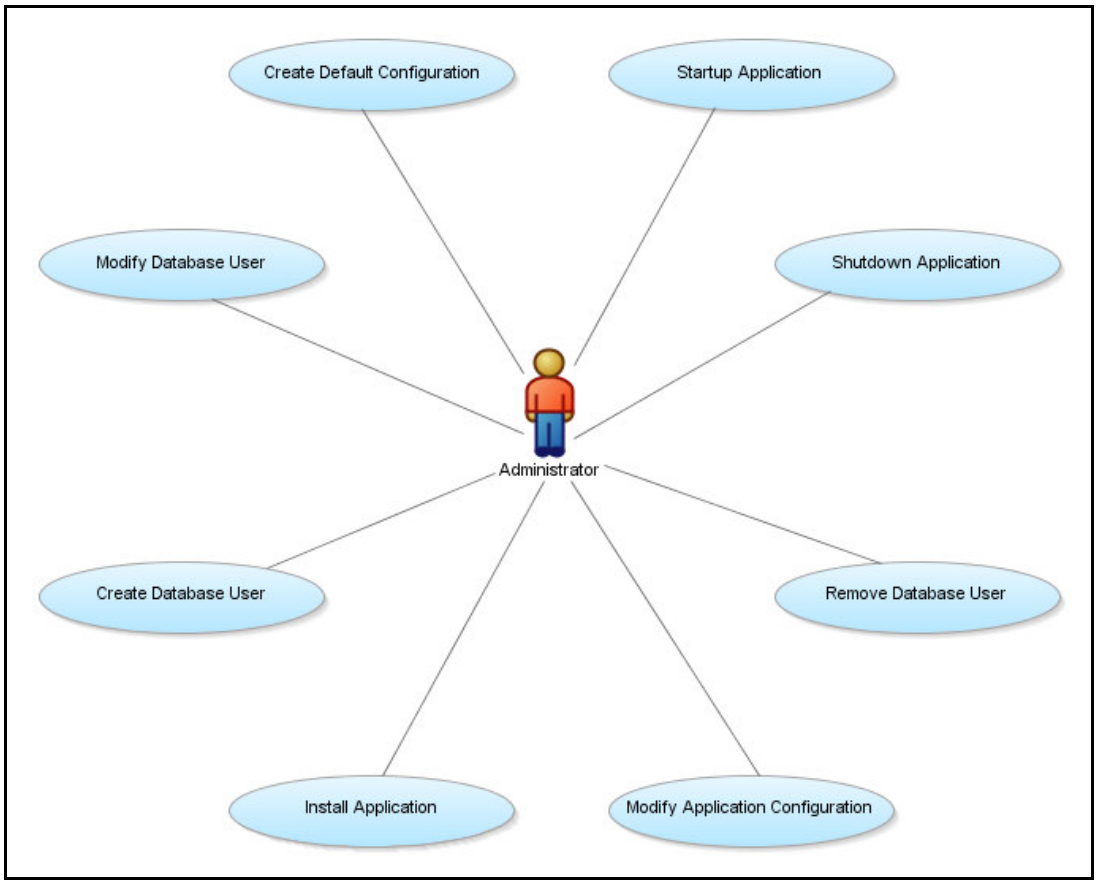

Figure B.1: Application Administration Diagram

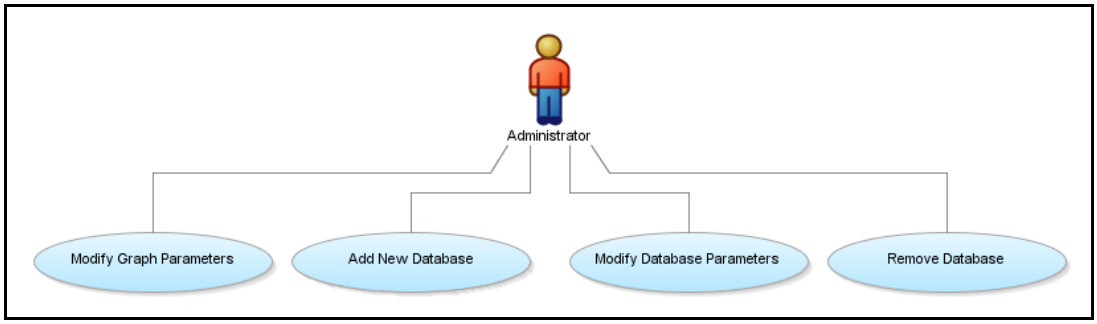

Figure B.2: System Configuration Diagram

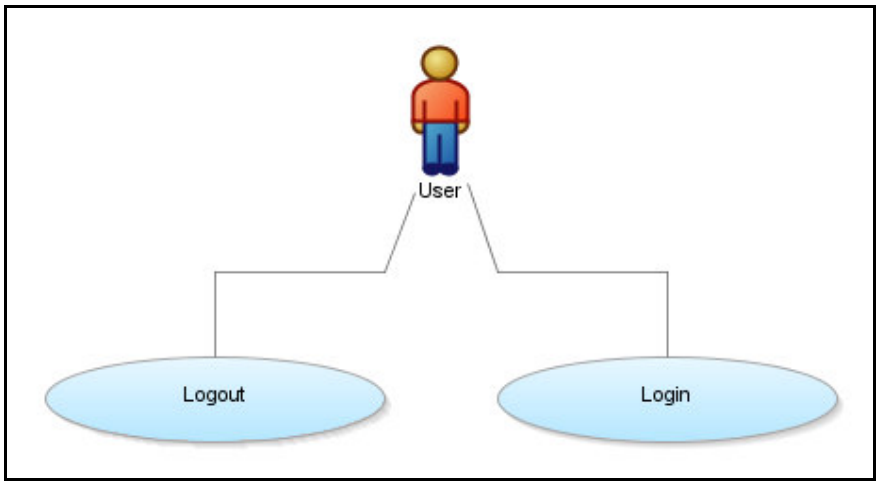

Figure B.3: User Authentication Diagram

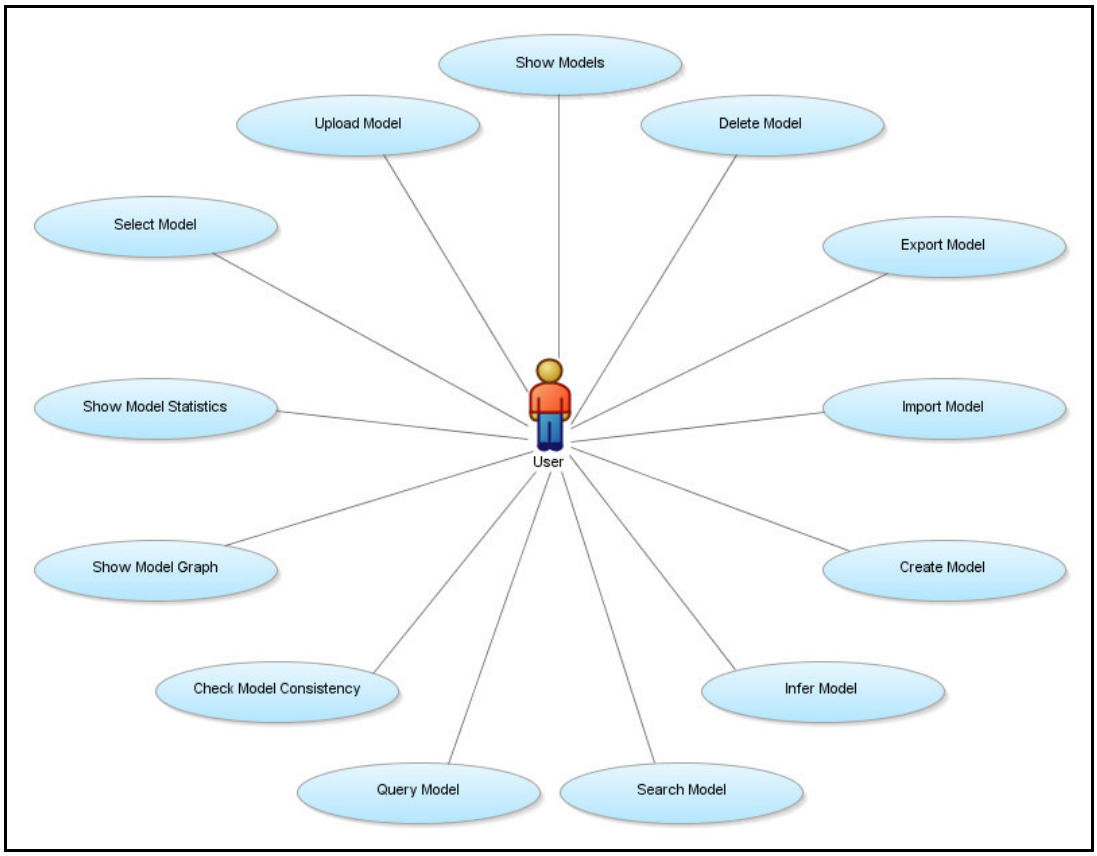

Figure B.4: Model Diagram

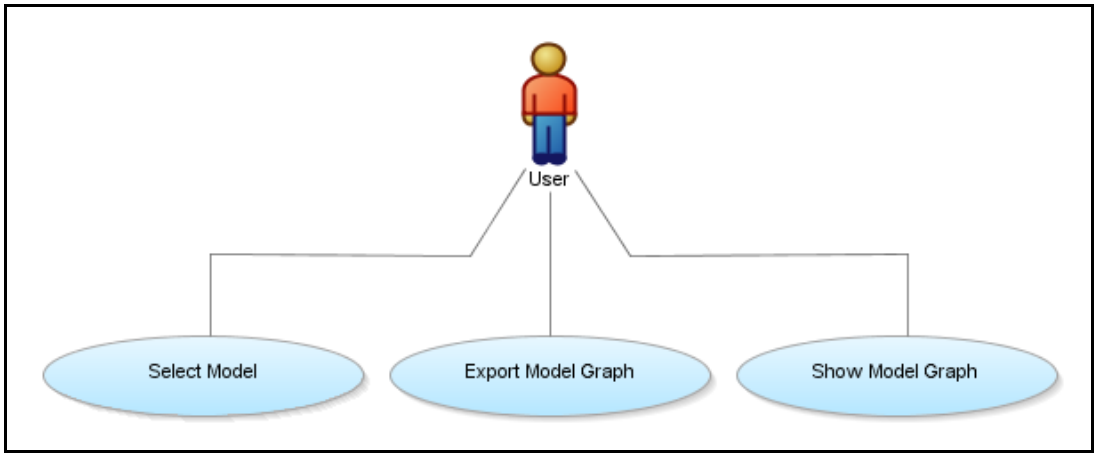

Figure B.5: Model Graph Diagram

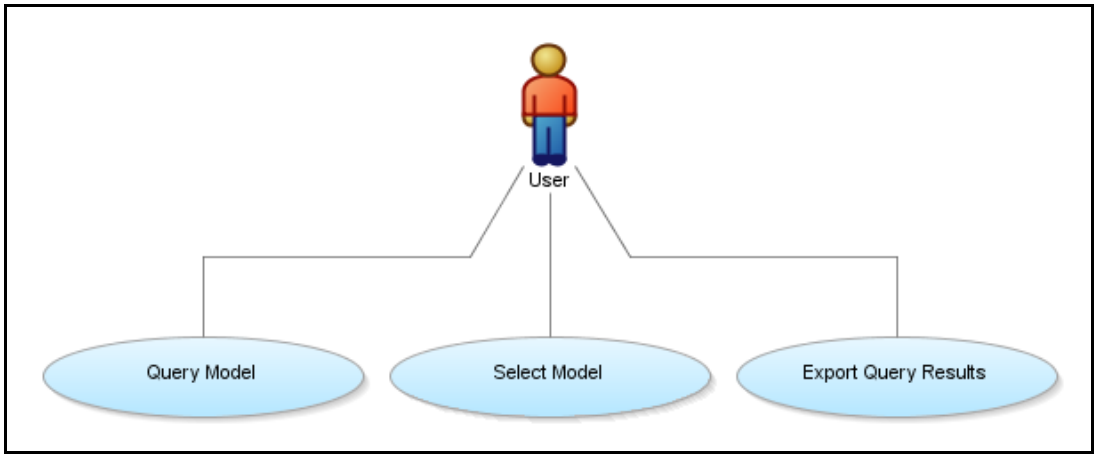

Figure B.6: Model Query Diagram

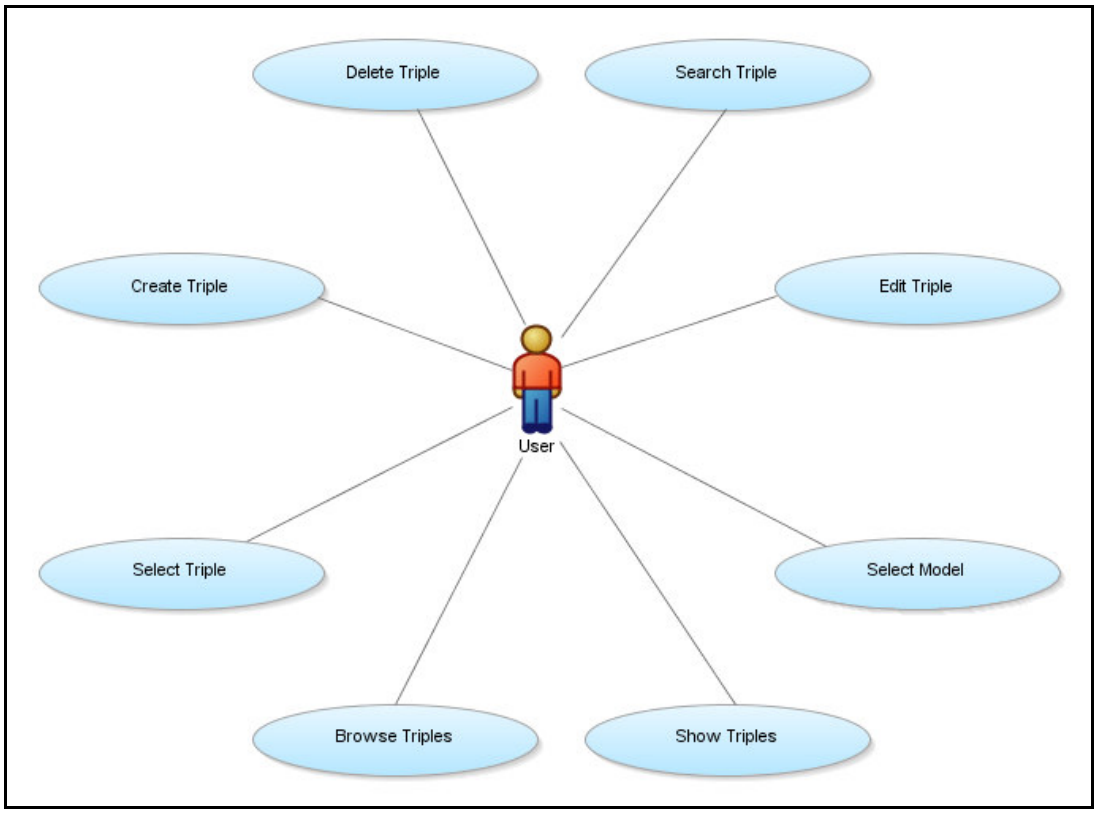

Figure B.7: Triple Diagram

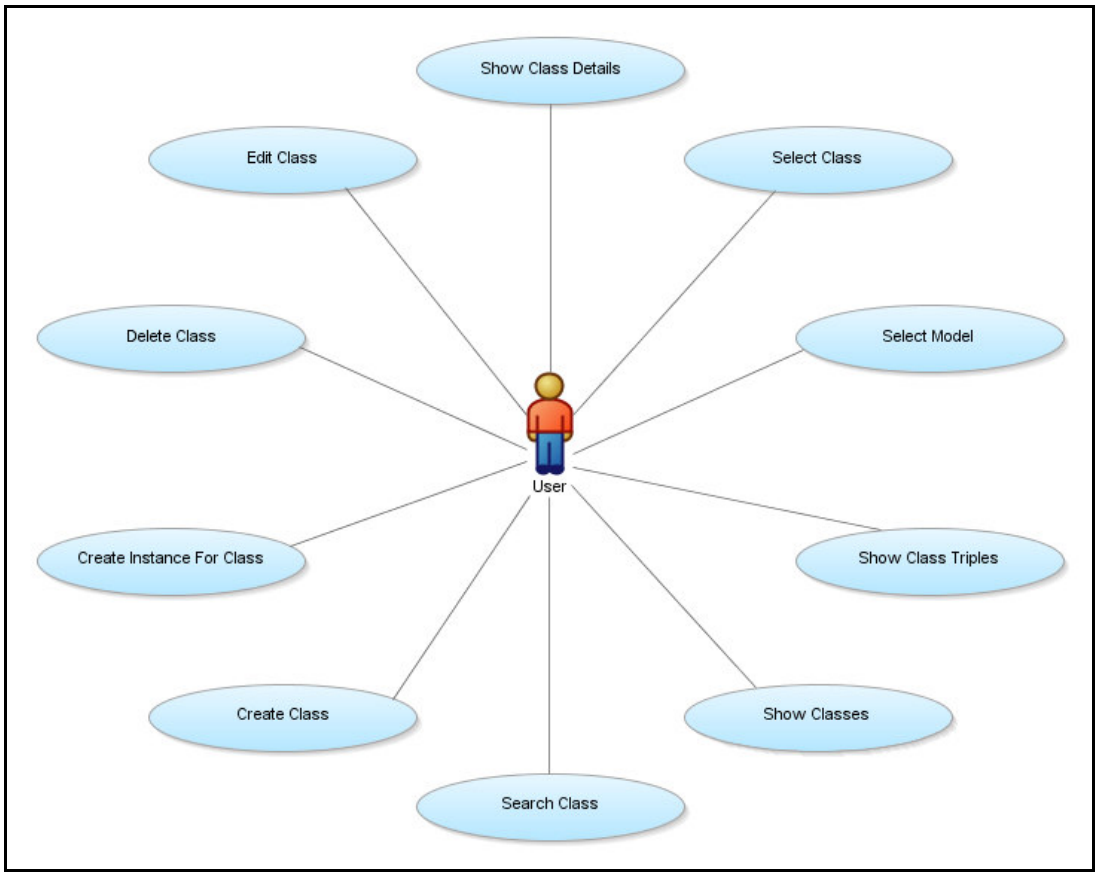

Figure B.8: Class Diagram

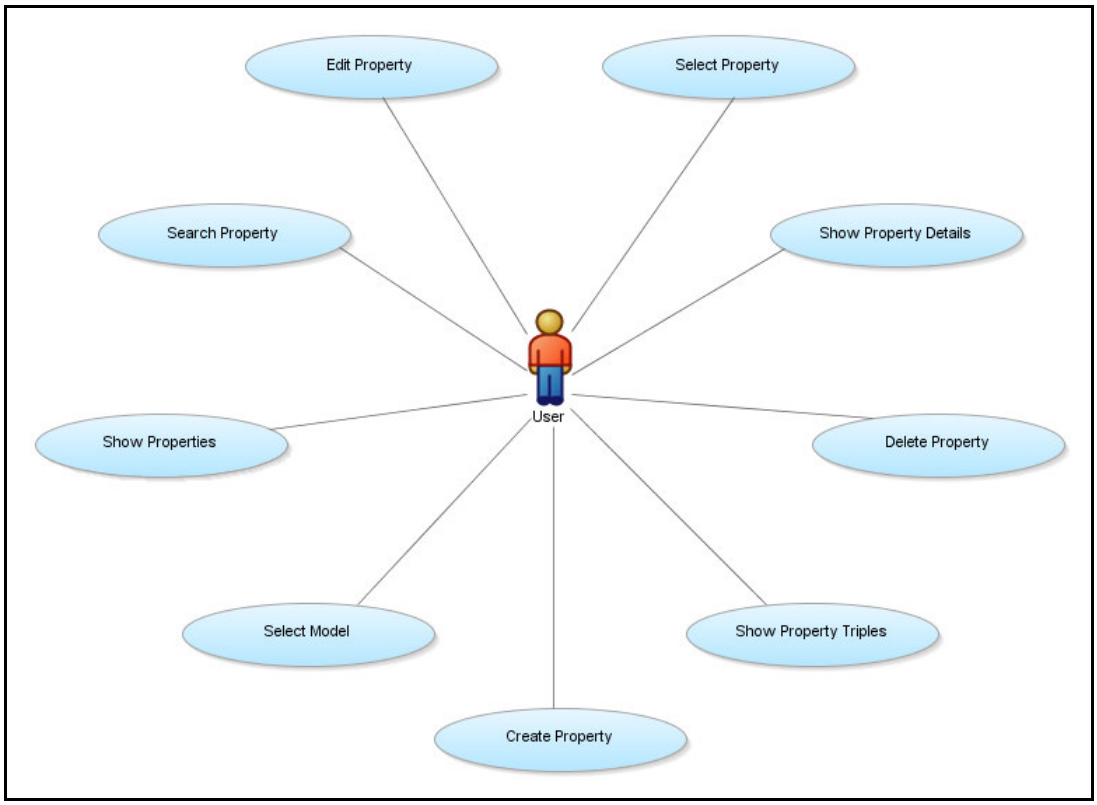

Figure B.9: Property Diagram

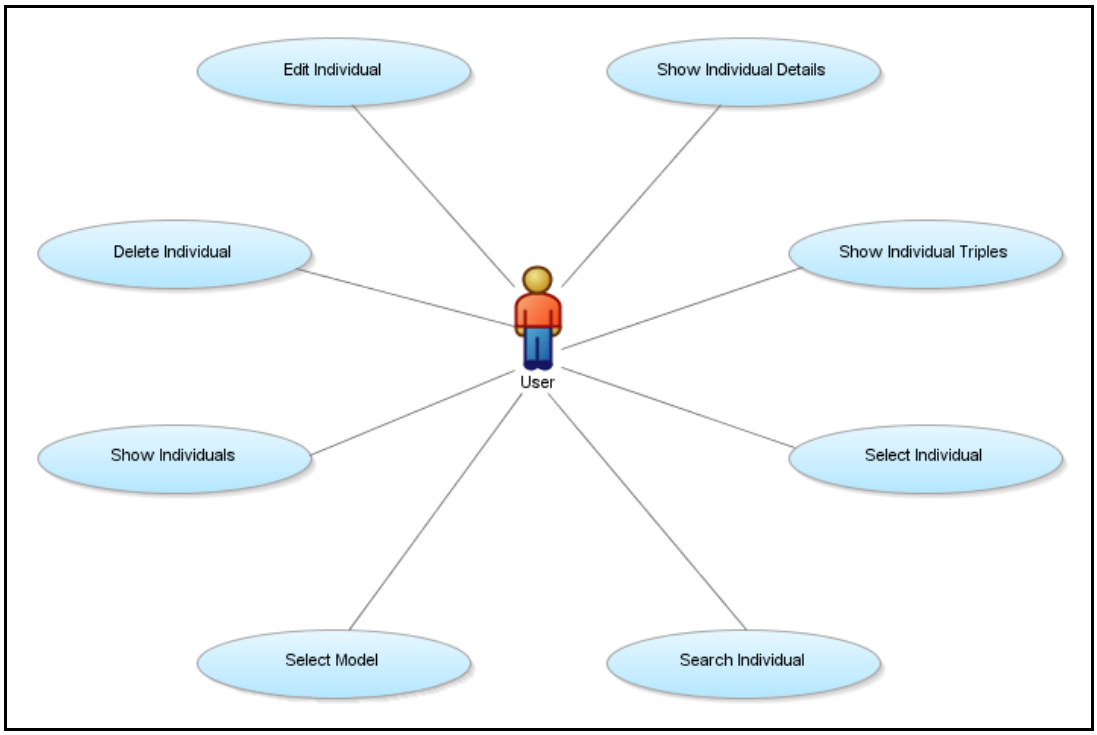

Figure B.10: Individual Diagram

B.3 Use Case Specifications

The following tables show the use case specifications of UltimateOMS.

Use Case Install Application
Stakeholder
Administrator
Goal of Primary Actor
Installing the application
Preconditions
Java Platform, Standard Edition (J2SE 1.5) is already installed and configured in server. An Application Server such as Tomcat, JBoss, Oracle internet Application Server (Oracle iAS), and so on is already installed and configured in server.
Scenario
Administrator deploys application's EAR or WAR file into the installed Application Server using provided tools in Application Server. In order to have graph facilities in application, Administrator also has to install Graphviz in server.

Table B.1: Use Case Install Application

Use Case Create Default Configuration
Stakeholder
Administrator
Goal of Primary Actor
Creating default configuration file which is used by application
Preconditions
Application is already installed in server. Application is shut down.
Scenario

Administrator creates default configuration file which is used by application. To do so, Administrator has to create systemConfig.xml. It includes some parameters related to databases and graph. By default when application is deployed, systemConfig.xml is automatically created and Administrator has to modify it.

Table B.2: Use Case Create Default Configuration

Use Case Modify Application Configuration
Stakeholder
Administrator
Goal of Primary Actor
Modifying default configuration file which is used by application
Preconditions
Application is already installed in server. Application is shut down.
Scenario
Administrator modifies configuration file, i.e. systemConfig.xml which is used by application. It includes some parameters related to databases and graph. Administrator modifies the configuration file for the following action: Adding new database Modifying database parameters Removing database Modifying graph parameters

Table B.3: Use Case Modify Application Configuration

Use Case Startup Application
Stakeholder
Administrator
Goal of Primary Actor
Starting up the application in order to use
Precondition
Application is already installed in server.
Scenario
Administrator starts up the installed Application Server in which the application has been deployed. By starting up the Application Server, the application automatically will be started up.

Table B.4: Use Case Startup Application

Use Case Shutdown Application
Stakeholder
Administrator
Goal of Primary Actor
Shutting down the application in order to modify configuration file or maintain database
Precondition
Application is already started up.
Scenario
Administrator shuts down the installed Application Server in which the application has been deployed. By shutting down the Application Server, the application automatically will be shut down.

Table B.5: Use Case Shutdown Application

Use Case Create Database User
Stakeholder
Administrator
Goal of Primary Actor
Creating database user
Preconditions
Database is already installed in server. Application is already installed in server.
Scenario
Administrator defines application's users by creating users in database. By using database administration's tool and logging in database as an admin role, Administrator could be able to create database's users. Created database's users are application's users and they can use application by logging in application through login page.

Table B.6: Use Case Create Database User

Use Case Modify Database User
Stakeholder
Administrator
Goal of Primary Actor
Modifying database user
Preconditions
Database is already installed in server. User is already created in database. Application is already installed in server.
Scenario
Administrator modifies application's users by modifying users in database. By using database administration's tool and logging in database as an admin role, Administrator could be able to modify database's users. Modified database's users are application's users and they can use application by logging in application through login page.

Table B.7: Use Case Modify Database User

Use Case Remove Database User
Stakeholder
Administrator
Goal of Primary Actor
Removing database user
Preconditions
Database is already installed in server. User is already created in database. Application is already installed in server.
Scenario
Administrator removes application's users by removing users in database. By using database administration's tool and logging in database as an admin role, Administrator could be able to remove database's users. Removed database's users are no longer application's users and they are not able to use application.

Table B.8: Use Case Remove Database User

Use Case Modify Graph Parameters
Stakeholder
Administrator
Goal of Primary Actor
Modifying graph parameters in configuration file
Precondition
Graphviz is already installed in server.
Scenario
Administrator modifies graph parameters in application's configuration file, i.e. systemConfig.xml. Graph parameters are: GRAPH_PANEL_WIDTH: The width of panel in which graph will be shown (in pixel). GRAPH_PANEL_HEIGHT: The height of panel in which graph will be shown (in pixel).

SERVLET_TMP_DIR:

Temporary directory path, which is used to save graph temporary files.

SERVLET_TMP_DIR˙ALIAS:

Alias which is defined in application server. It is a URL map in order to access the SERVLET˙TMP˙DIR contents by application server.

GRAPH_VIZ_ROOT:

Path of the Graphviz root directory.

GRAPH_VIZ_PATH:

Path of the dot.exe directory in Graphviz.

GRAPH_VIZ_FONT_DIR:

Path of the font directory in Graphviz.

Table B.9: Use Case Modify Graph Parameters

Use Case Add New Database
Stakeholder
Administrator
Goal of Primary Actor
Adding database parameters in configuration file
Precondition
New database is already installed and configured.
Scenario
Administrator adds new database parameters in application's configuration file, i.e. systemConfig.xml. Database parameters are: • **description**: brief description of the new database. • **id**: the name of database which is used in Jena. • **driver**: driver of the new database. • **url**: url of the new database. Different types of databases which are currently supported built in Jena 2.5.2, are as the following list:

MySQL
HSQLDB
Apache Derby
PostgreSQL
Oracle
Microsoft SQL Server

Table B.10: Use Case Add New Database

Use Case Modify Database Parameters
Stakeholder
Administrator
Goal of Primary Actor
Modifying database parameters in configuration file
Precondition
Database is already installed and configured.
Scenario
Administrator modifies database parameters in application's configuration file, i.e. systemConfig.xml. Database parameters are: • **description**: brief description of the new database. • **id**: the name of database which is used in Jena. • **driver**: driver of the new database. • **url**: url of the new database. Different types of databases which are currently supported built in Jena 2.5.2, are as the following list: MySQL HSQLDB Apache Derby PostgreSQL Oracle Microsoft SQL Server

Table B.11: Use Case Modify Database Parameters

Use Case Remove Database
Stakeholder
Administrator
Goal of Primary Actor
Removing database parameters in configuration file
Precondition
Database is already uninstalled or it is not longer supported.
Scenario
Administrator removes related database parameters in application's configuration file, i.e. systemConfig.xml. Database parameters are: • **description**: brief description of the new database. • **id**: the name of database which is used in Jena. • **driver**: driver of the new database. • **url**: url of the new database.

Table B.12: Use Case Remove Database

Use Case Login
Stakeholder
User
Goal of Primary Actor
Login to the application in order to manage semantic data
Preconditions
Application is running. Database is already started up.
Scenario
User can login into different databases, which are provided by Administrator. By receiving username, password, and selected database , application will check if the User is a valid user in selected database. If succeed, a new session will be created to the User and application's main page will be forwarded to the User. If not, User will be informed the reason of login failure through returned message from the application.

Table B.13: Use Case Login

Use Case Logout
Stakeholder
User
Goal of Primary Actor
Logout from the application
Preconditions
Application is running. Database is already started up. User has already logged in the application.
Scenario
User can logout from application by sending a request to the application. In this case, application will destroy User's session and application's login page will be forwarded to the User.

Table B.14: Use Case Logout

Use Case Show Models
Stakeholder
User
Goal of Primary Actor
Viewing list of models stored in the connected database
Preconditions
Application is running. Database is already started up. User has already logged in the application.
Scenario
User can see list of models stored in the connected database by sending a request to the application. Application will return all the existing models stored in the connected database as a list to the User.

Table B.15: Use Case Show Models

Use Case Create Model
Stakeholder
User
Goal of Primary Actor
Creating an empty model in the connected database
Preconditions
Application is running. Database is already started up. User has already logged in the application.
Scenario
User can create an empty model in the connected database by providing all necessary information in the "model create form". By receiving provided information for model, application will go through model creation's business logics and create it if there was no wrong data or unexpected exception. User will be informed the result of operation through returned message from the application.

Table B.16: Use Case Create Model

Use Case Import Model
Stakeholder
User
Goal of Primary Actor
Importing a model from anywhere in Internet into the connected database
Preconditions
Application is running. Database is already started up. User has already logged in the application.
Scenario
User can import a model from anywhere in Internet into the connected database by providing all necessary information in the "model import form". By receiving provided information for importing model, application will go through model importation's business logics and

import it from specified URL if there was no wrong data or unexpected exception. User will be informed the result of operation through returned message from the application.

The following file formats can be imported into the connected database:
RDF/XML
N-Triples
Notation3/N3

Table B.17: Use Case Import Model

Use Case Upload Model
Stakeholder
User
Goal of Primary Actor
Uploading a model from local file system into the connected database
Preconditions
Application is running. Database is already started up. User has already logged in the application.
Scenario
User can upload a model from local file system into the connected database by providing all necessary information in the "model upload form". By receiving provided information for uploading model, application will go through model uploading's business logics and upload it from specified local directory path if there was no wrong data or unexpected exception. User will be informed the result of operation through returned message from the application. The following file formats can be uploaded into the connected database: RDF/XML N-Triples Notation3/N3

Table B.18: Use Case Upload Model

Use Case Search Model
Stakeholder
User
Goal of Primary Actor
Searching for specific models stored in the connected database
Preconditions
Application is running. Database is already started up. User has already logged in the application.
Scenario
User can search for a specific list of models stored in the connected database by typing some part of desired models' name and sending a request to the application. Application will return all the filtered models, stored in the connected database, as a list to the User in which models' names contain the User typed part.

Table B.19: Use Case Search Model

Use Case Select Model
Stakeholder
User
Goal of Primary Actor
Selecting a default model from the list of existing models in database
Preconditions
Application is running. Database is already started up. User has already logged in the application. There is at least one model already existing in database.
Scenario
User selects a default model from the list of existing models in database in order to deal with it.

Table B.20: Use Case Select Model

Use Case Export Model
Stakeholder
User
Goal of Primary Actor
Exporting a selected model from the connected database into the local file system
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has been selected.
Scenario
User can export the default model from the connected database into local file system by providing all necessary information in the "model export form". By receiving provided information for exporting model, application will go through model exportation's business logics and export it from connected database if there was no wrong data or unexpected exception. User will be asked to specify a local directory path in which the exported model will be saved. The default model can be exported as following formats: RDF/XML N-Triples Notation3/N3

Table B.21: Use Case Export Model

Use Case Show Model Graph
Stakeholder
User
Goal of Primary Actor
Viewing graph of a selected model
Preconditions
Application is running. Database is already started up. Graphviz is already installed. User has already logged in the application. A default model has been selected.
Scenario
User can see graph of the default model by sending a request to the application. Application will parse the default model and generate a proper input file for Graphviz. By sending generated input file to the Graphviz and executing it, applicati on will receive the generated output file from Graphviz. The received file by application will be input file for a java applet application, which will be sent to the User as a graph. Therefore, application will return the graph of the default model to the User through loading java applet.

Table B.22: Use Case Show Model Graph

Use Case Export Model Graph
Stakeholder
User
Goal of Primary Actor
Exporting graph of a selected model from the connected database into the local file system
Preconditions
Application is running. Database is already started up. Graphviz is already installed. User has already logged in the application.

A default model has been selected.

Scenario

User can export graph of the default model from the connected database into local file system by providing all necessary information in the "model's graph export form". By receiving provided information for exporting model's graph, application will go through model's graph exportation's business logics and export it if there was no wrong data or unexpected exception. User will be asked to specify a local directory path in which the exported graph of model will be saved.

The default model's graph can be exported as following formats:
PNG (Portable Network Graphics)
SVG (Scalable Vector Graphics)
GIF (Graphics Interchange Format)
PostScript

Table B.23: Use Case Export Model Graph

Use Case Query Model
Stakeholder
User
Goal of Primary Actor
Querying a selected model from the connected database
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has been selected.
Scenario
User can query the default model from the connected database by providing desired SPARQL query in the "model query form". By receiving provided information for querying model, application will go through model query's business logics and query it if there was no wrong data or unexpected exception. Application will return all

the query results as a list to the User.

Table B.24: Use Case Query Model

Use Case Export Query Results
Stakeholder
User
Goal of Primary Actor
Exporting query results of a selected model from the connected database into the local file system
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has been selected.
Scenario
User can export query results of the default model from the connected database into local file system by providing all necessary information in the "model's query results export form". By receiving provided information for exporting model's query results, application will go through model's query results exportation's business logics and export it if there was no wrong data or unexpected exception. User will be asked to specify a local directory path in which the exported query results of model will be saved. The default model's query results can be exported as following formats: N-Triples Notation3/N3 RDF/XML Text File XML File

Table B.25: Use Case Export Query Results

Use Case Infer Model
Stakeholder
User
Goal of Primary Actor
Inferring a selected model from the connected database
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has been selected.
Scenario
User can infer the default model from the connected database by providing all necessary information in the "model infer form". By receiving provided information for inferring model, application will go through model inferring's business logics and infer it, using different types of "Reasoner" or "Generic Rule Reasoner", if there was no wrong data or unexpected exception. The inferred results will be saved in connected database as a new model with a name, which is specified by User in "model infer form". User will be informed the result of operation through returned message from the application. Different types of reasoner, which are currently supported built in Jena 2.5.2, are as the following list: Generic Rule Reasoner OWL Micro Reasoner OWL Mini Reasoner OWL Reasoner RDFS Default Reasoner RDFS Full Reasoner RDFS Simple Reasoner Transitive Reasoner

Table B.26: Use Case Infer Model

Use Case Show Model Statistics
Stakeholder
User
Goal of Primary Actor
Viewing statistics of a selected model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has been selected.
Scenario
User can see statistics of the default model by sending a request to the application. Application will return all statistics information of the default model to the User.

Table B.27: Use Case Show Model Statistics

Use Case Check Model Consistency
Stakeholder
User
Goal of Primary Actor
Checking a selected model consistency in the connected database
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has been selected.
Scenario
User can check the default model consistency in the connected database by providing all necessary information in the "check model consistency form". By receiving provided information for checking model consistency, application will go through model consistency checking's business logics and generate a report if there was no unexpected exception. User will be informed the result of operation through

returned message from the application.

Table B.28: Use Case Check Model Consistency

Use Case Delete Model
Stakeholder
User
Goal of Primary Actor
Deleting a selected model from the connected database
Preconditions
Application is running. Database is already started up. User has already logged in the application. One of the models of connected database has been selected.
Scenario
User can delete a selected model by sending a request to the application. Application will delete selected model from the connected database. User will be informed the result of operation through returned message from the application.

Table B.29: Use Case Delete Model

Use Case Show Triples
Stakeholder
User
Goal of Primary Actor
Viewing list of triples of default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected.
Scenario

User can see list of triples of the default model by sending a request to the application. Application will return all the existing triples of the default model as a list to the User.

Table B.30: Use Case Show Triples

Use Case Create Triple
Stakeholder
User
Goal of Primary Actor
Creating a triple for default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected.
Scenario
User can create a triple for default model by providing all necessary information in the "triple create form". By receiving provided information for triple, application will go through triple creation's business logics and create it if there was no wrong data or unexpected exception. User will be informed the result of operation through returned message from the application.

Table B.31: Use Case Create Triple

Use Case Search Triple
Stakeholder
User
Goal of Primary Actor
Searching for specific triples of default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected.
Scenario
User can search for a specific list of triples of default model by typing some part of desired triples' subject name, predicate name, object name, or all of them and sending a request to the application. Application will return all the filtered triples of the default model as a list to the User in which triples' subject name, predicate name, object name, or all of them contain the User typed part.

Table B.32: Use Case Search Triple

Use Case Browse Triples
Stakeholder
User
Goal of Primary Actor
Browsing triples of a selected resource in default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. A resource in default model has been selected.
Scenario
User can browse triples of a selected resource in default model by sending a request to the application. Application will return all the filtered triples of the default model as a three different list to the

User in which triples' subject name, predicate name, or object name is equal to the selected resource.

Table B.33: Use Case Browse Triples

Use Case Select Triple
Stakeholder
User
Goal of Primary Actor
Selecting a triple from the list of existing triples in default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. There is at least one triple already existing in default model.
Scenario
User selects a triple from the list of existing triples in default model in order to deal with it.

Table B.34: Use Case Select Triple

Use Case Edit Triple
Stakeholder
User
Goal of Primary Actor
Editing a selected triple of default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the triples of default model has been selected.

Scenario
User can edit a selected triple of default model by modifying all necessary information in the “triple edit form”. By receiving modified information for triple, application will go through triple modification’s business logics and modify it if there was no wrong data or unexpected exception. User will be informed the result of operation through returned message from the application.

Table B.35: Use Case Edit Triple

Use Case Delete Triple
Stakeholder
User
Goal of Primary Actor
Deleting a selected triple from default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the triples of default model has been selected.
Scenario
User can delete a selected triple by sending a request to the application. Application will delete a selected triple from default model. User will be informed the result of operation through returned message from the application.

Table B.36: Use Case Delete Triple

Use Case Show Classes
Stakeholder
User
Goal of Primary Actor
Viewing list of classes of default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected.
Scenario
User can see list of classes of the default model by sending a request to the application. Application will return all the existing classes of the default model as a list to the User.

Table B.37: Use Case Show Classes

Use Case Create Class
Stakeholder
User
Goal of Primary Actor
Creating a class for default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected.
Scenario
User can create a class for default model by providing all necessary information in the "class create form". By receiving provided information for class, application will go through class creation's business logics and create it if there was no wrong data or unexpected exception. User will be informed the result of operation through returned message from the application.

Table B.38: Use Case Create Class

Use Case Search Class
Stakeholder
User
Goal of Primary Actor
Searching for specific classes of default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected.
Scenario
User can search for a specific list of classes of default model by typing some part of desired classes' name and sending a request to the application. Application will return all the filtered classes of the default model as a list to the User in which classes' names contain the User typed part.

Table B.39: Use Case Search Class

Use Case Select Class
Stakeholder
User
Goal of Primary Actor
Selecting a class from the list of existing classes in default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. There is at least one class already existing in default model.
Scenario
User selects a class from the list of existing classes in default model in order to deal with it.

Table B.40: Use Case Select Class

Use Case Show Class Details
Stakeholder
User
Goal of Primary Actor
Viewing details information of a selected class
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the classes of default model has been selected.
Scenario
User can see detail information of a selected class by sending a request to the application. Application will return all categorized detail information of the selected class to the User.

Table B.41: Use Case Show Class Details

Use Case Show Class Triples
Stakeholder
User
Goal of Primary Actor
Viewing list of triples of a selected class
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the classes of default model has been selected.
Scenario
User can see list of triples of a selected class by sending a request to the application. Application will return all the existing triples of the selected class as a list to the User.

Table B.42: Use Case Show Class Triples

Use Case Create Instance For Class
Stakeholder
User
Goal of Primary Actor
Creating instance for a selected class
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the classes of default model has been selected.
Scenario
User can create an instance for a selected class by providing all necessary information in the "instance create form". By receiving provided information for instance, application will go through instance creation's business logics and create it if there was no wrong data or unexpected exception. User will be informed the result of operation through returned message from the application.

Table B.43: Use Case Create Instance For Class

Use Case Edit Class
Stakeholder
User
Goal of Primary Actor
Editing a selected class of default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the classes of default model has been selected.
Scenario

User can edit a selected class of default model by modifying all necessary information in the “class edit form”. By receiving modified information for class, application will go through class modification’s business logics and modify it if there was no wrong data or unexpected exception. User will be informed the result of operation through returned message from the application.

Table B.44: Use Case Edit Class

Use Case Delete Class
Stakeholder
User
Goal of Primary Actor
Deleting a selected class from default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the classes of default model has been selected.
Scenario
User can delete a selected class by sending a request to the application. Application will delete selected class from default model. User will be informed the result of operation through returned message from the application.

Table B.45: Use Case Delete Class

Use Case Show Properties
Stakeholder
User
Goal of Primary Actor
Viewing list of properties of default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected.
Scenario
User can see list of properties of the default model by sending a request to the application. Application will return all the existing properties of the default model as a list to the User.

Table B.46: Use Case Show Properties

Use Case Create Property
Stakeholder
User
Goal of Primary Actor
Creating a property for default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected.
Scenario
User can create a property for default model by providing all necessary information in the "property create form". By receiving provided information for property, application will go through property creation's business logics and create it if there was no wrong data or unexpected exception. User will be informed the result of operation through returned message from the application.

Table B.47: Use Case Create Property

Use Case Search Property
Stakeholder
User
Goal of Primary Actor
Searching for specific properties of default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected.
Scenario
User can search for a specific list of properties of default model by typing some part of desired properties' name and sending a request to the application. Application will return all the filtered properties of the default model as a list to the User in which properties' names contain the User typed part.

Table B.48: Use Case Search Property

Use Case Select Property
Stakeholder
User
Goal of Primary Actor
Selecting a property from the list of existing properties in default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. There is at least one property already existing in default model.
Scenario
User selects a property from the list of existing properties in default model in order to deal with it.

Table B.49: Use Case Select Property

Use Case Show Property Details
Stakeholder
User
Goal of Primary Actor
Viewing details information of a selected property
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the properties of default model has been selected.
Scenario
User can see detail information of a selected property by sending a request to the application. Application will return all categorized detail information of the selected property to the User.

Table B.50: Use Case Show Property Details

Use Case Show Property Triples
Stakeholder
User
Goal of Primary Actor
Viewing list of triples of a selected property
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the properties of default model has been selected.
Scenario
User can see list of triples of a selected property by sending a request to the application. Application will return all the existing triples of the selected property as a list to the User.

Table B.51: Use Case Show Property Triples

Use Case Edit Property
Stakeholder
User
Goal of Primary Actor
Editing a selected property of default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the properties of default model has been selected.
Scenario
User can edit a selected property of default model by modifying all necessary information in the "property edit form". By receiving modified information for property, application will go through property modification's business logics and modify it if there was no wrong data or unexpected exception. User will be informed the result of operation through returned message from the application.

Table B.52: Use Case Edit Property

Use Case Delete Property
Stakeholder
User
Goal of Primary Actor
Deleting a selected property from default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the properties of default model has been selected.
Scenario

User can delete a selected property by sending a request to the application. Application will delete selected property from default model. User will be informed the result of operation through returned message from the application.

Table B.53: Use Case Delete Property

Use Case Show Individuals
Stakeholder
User
Goal of Primary Actor
Viewing list of individuals of default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected.
Scenario
User can see list of individuals of the default model by sending a request to the application. Application will return all the existing individuals of the default model as a list to the User.

Table B.54: Use Case Show Individuals

Use Case Search Individual
Stakeholder
User
Goal of Primary Actor
Searching for specific individuals of default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected.

Scenario
User can search for a specific list of individuals of default model by typing some part of desired individuals' name and sending a request to the application. Application will return all the filtered individuals of the default model as a list to the User in which individuals' names contain the User typed part.

Table B.55: Use Case Search Individual

Use Case Select Individual
Stakeholder
User
Goal of Primary Actor
Selecting an individual from the list of existing individuals in default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. There is at least one individual already existing in default model.
Scenario
User selects an individual from the list of existing individuals in default model in order to deal with it.

Table B.56: Use Case Select Individual

Use Case Show Individual Details
Stakeholder
User
Goal of Primary Actor
Viewing details information of a selected individual
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the individuals of default model has been selected.
Scenario
User can see detail information of a selected individual by sending a request to the application. Application will return all categorized detail information of the selected individual to the User.

Table B.57: Use Case Show Individual Details

Use Case Show Individual Triples
Stakeholder
User
Goal of Primary Actor
Viewing list of triples of a selected individual
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the individuals of default model has been selected.
Scenario
User can see list of triples of a selected individual by sending a request to the application. Application will return all the existing triples of the selected individual as a list to the User.

Table B.58: Use Case Show Individual Triples

Use Case Edit Individual
Stakeholder
User
Goal of Primary Actor
Editing a selected individual of default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the individuals of default model has been selected.
Scenario
User can edit a selected individual of default model by modifying all necessary information in the "individual edit form". By receiving modified information for individual, application will go through individual modification's business logics and modify it if there was no wrong data or unexpected exception. User will be informed the result of operation through returned message from the application.

Table B.59: Use Case Edit Individual

Use Case Delete Individual
Stakeholder
User
Goal of Primary Actor
Deleting a selected individual from default model
Preconditions
Application is running. Database is already started up. User has already logged in the application. A default model has already been selected. One of the individuals of default model has been selected.
Scenario
User can delete a selected individual by sending a request to the application. Application will delete selected individual from default model. User will be informed the result of operation through returned message from the application.

Table B.60: Use Case Delete Individual

C - UltimateOMS Screenshots

The following figures show some screenshots of UltimateOMS.

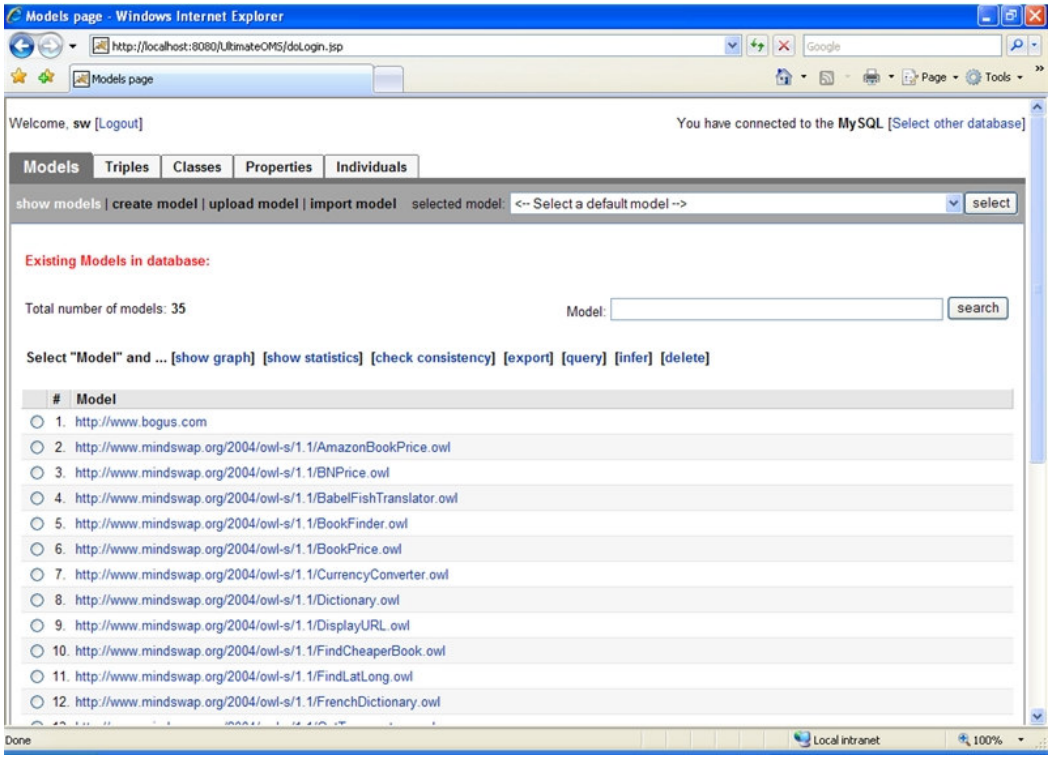

Figure C.1: UltimateOMS Models page

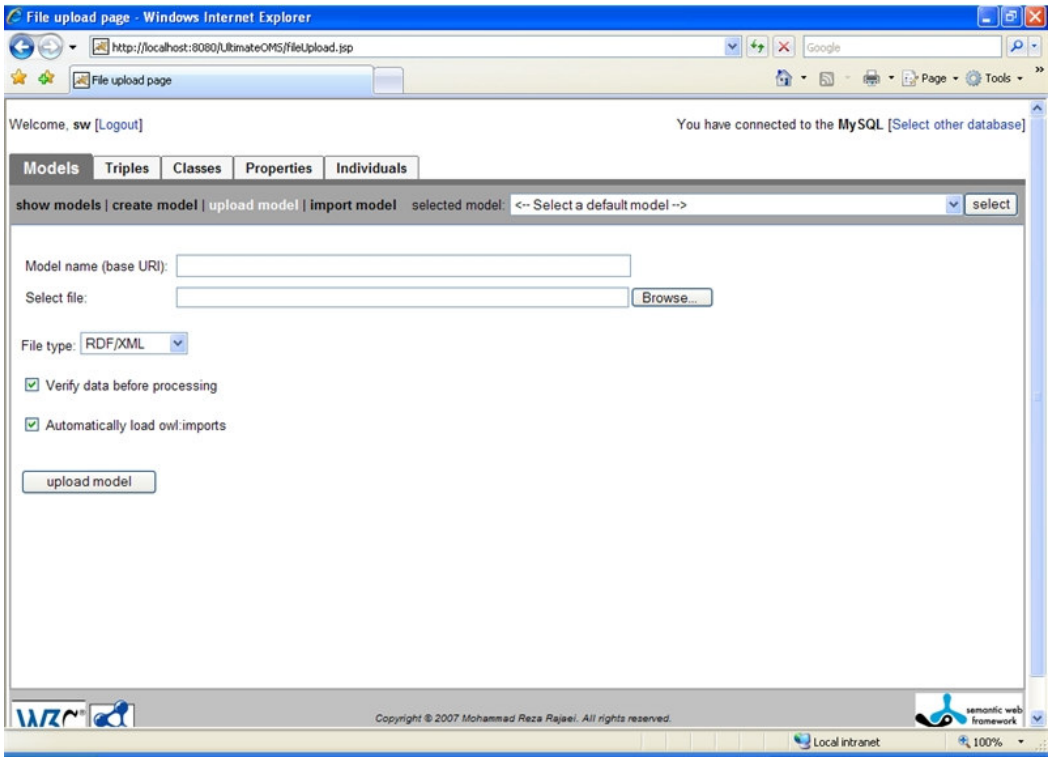

Figure C.2: Uploading model page in UltimateOMS

Figure C.3: Visualization graph in UltimateOMS

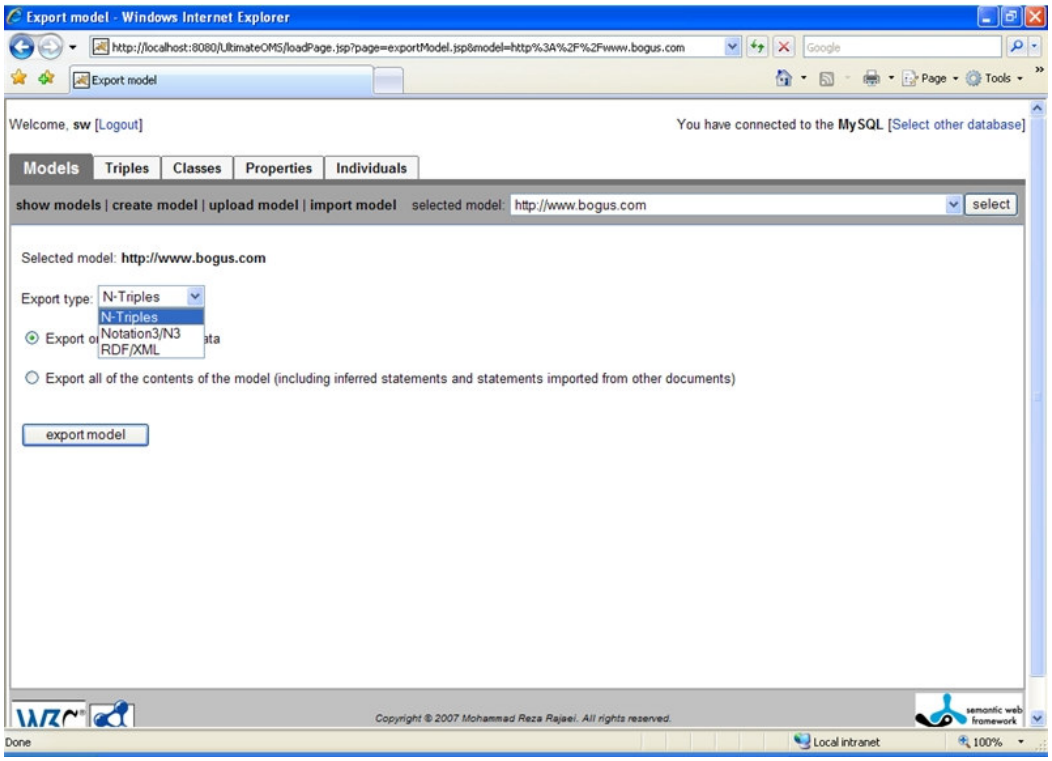

Figure C.4: Exporting model page in UltimateOMS

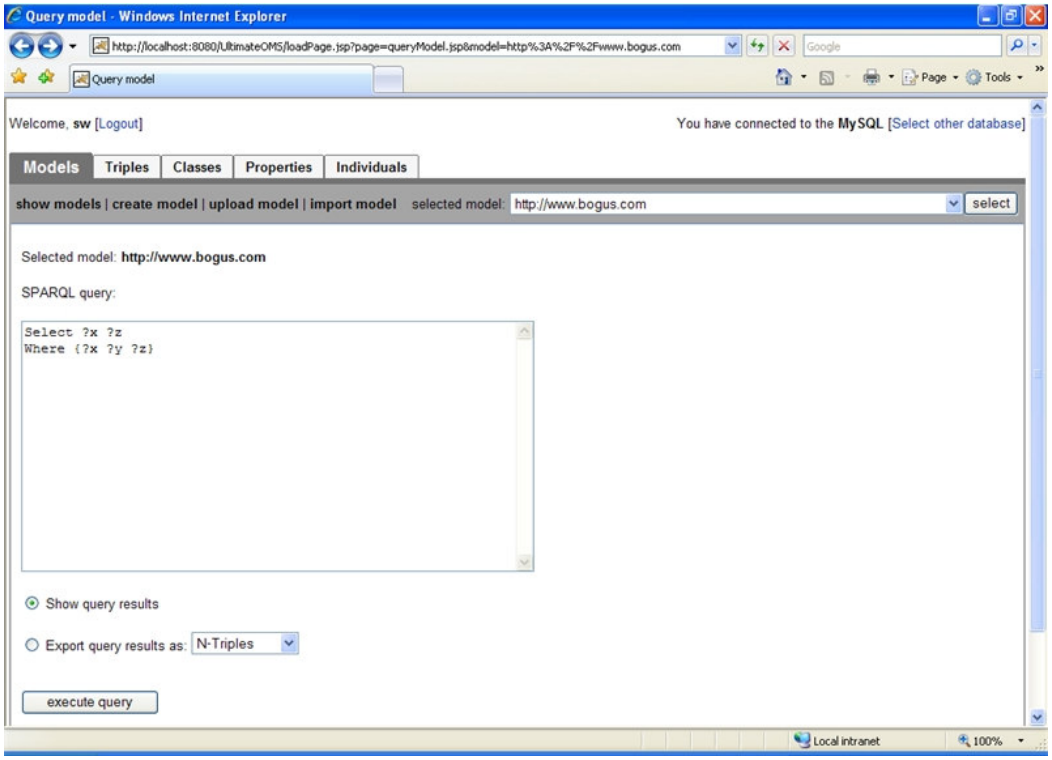

Figure C.5: Querying model page in UltimateOMS

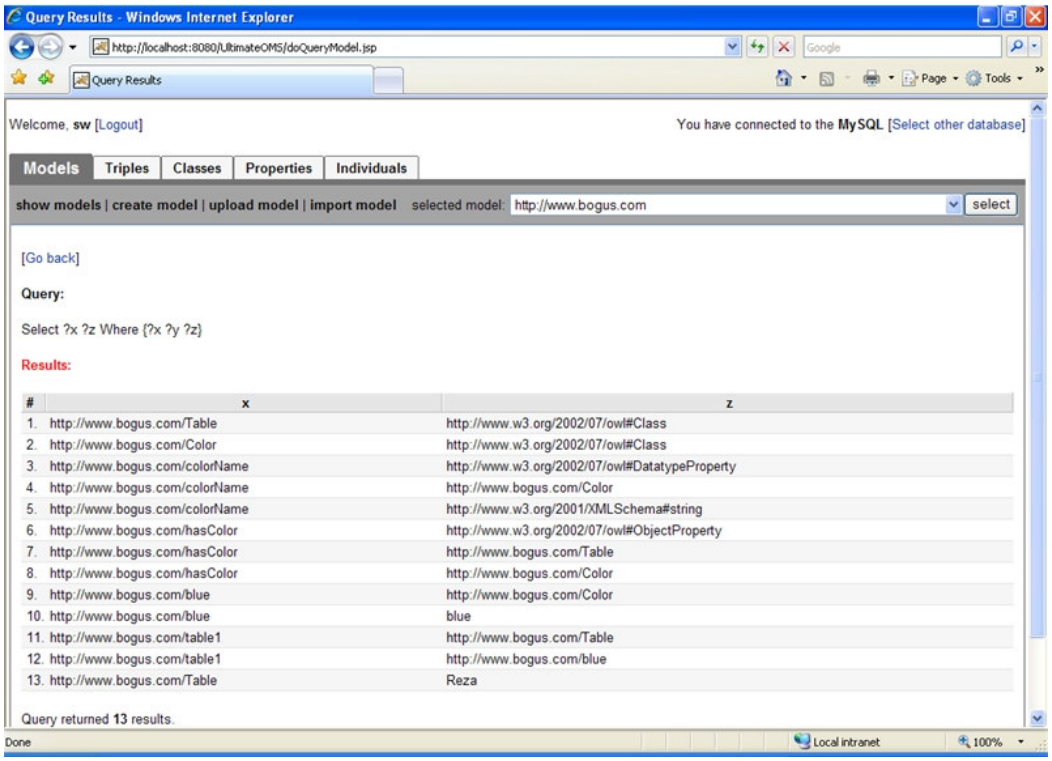

Figure C.6: Model query result page in UltimateOMS

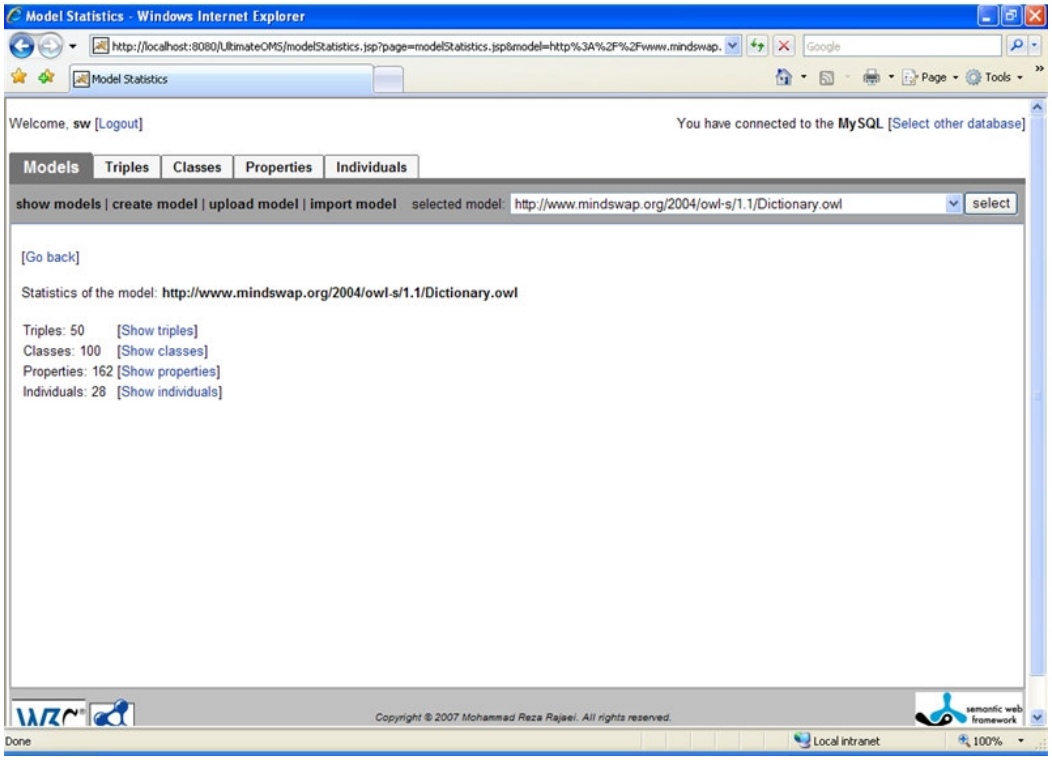

Figure C.7: Model statistics page in UltimateOMS

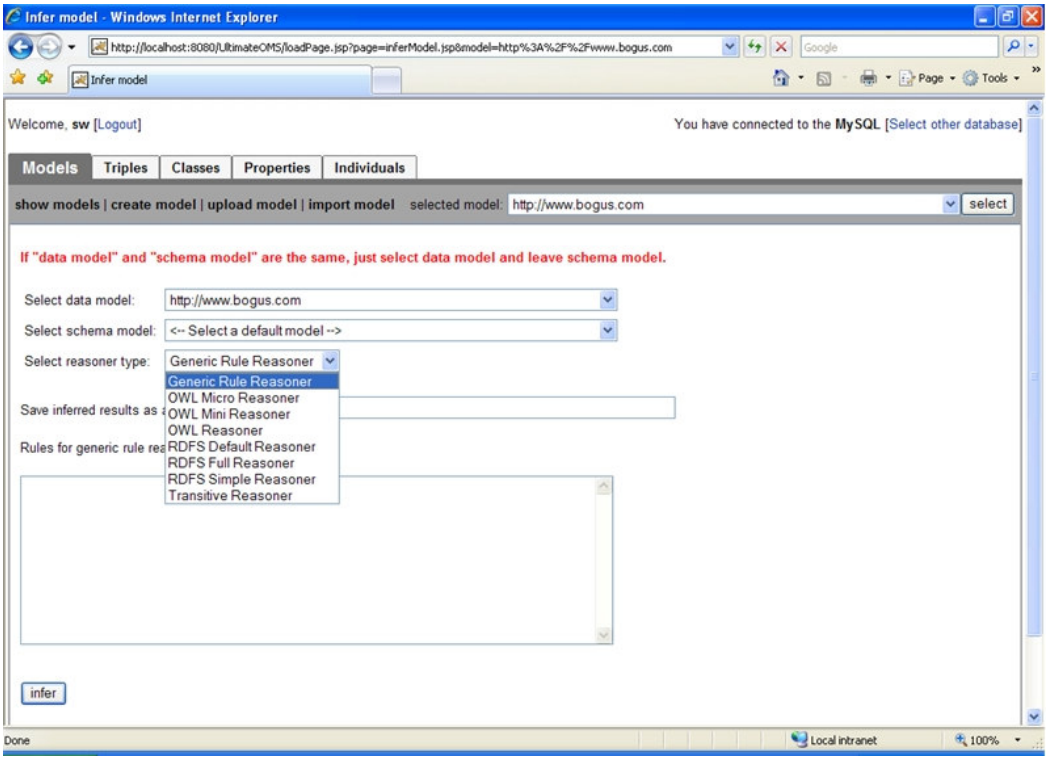

Figure C.8: Inferring model page in UltimateOMS

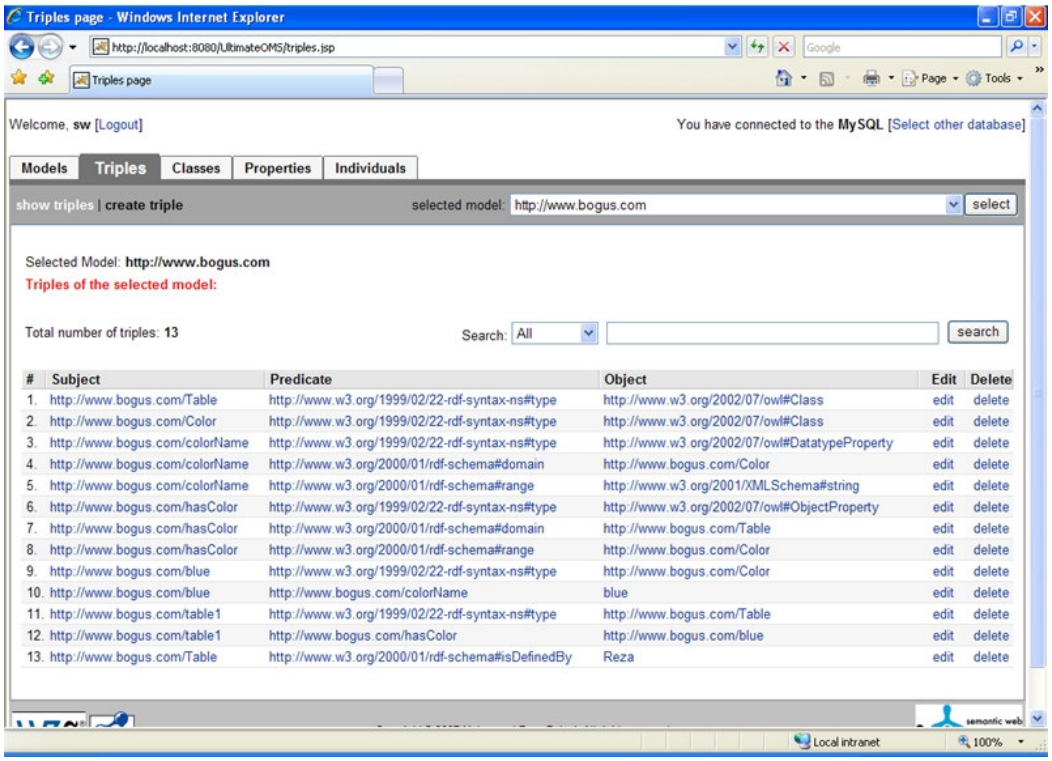

Figure C.9: UltimateOMS Triples page

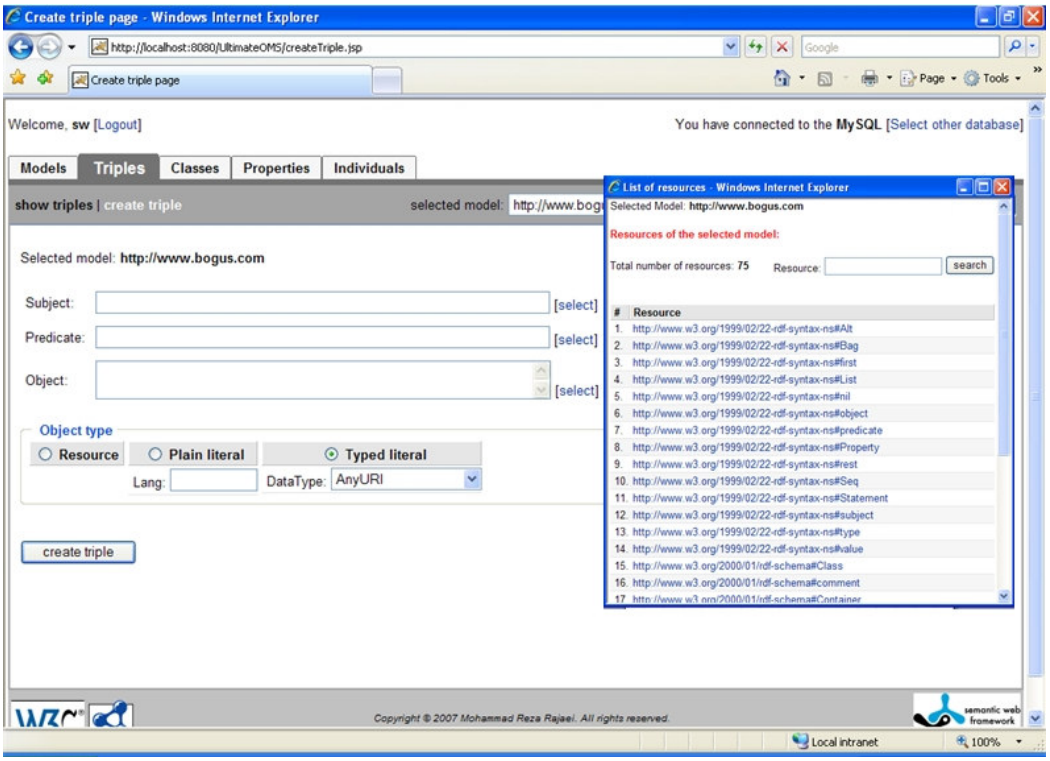

Figure C.10: Creating triple page in UltimateOMS

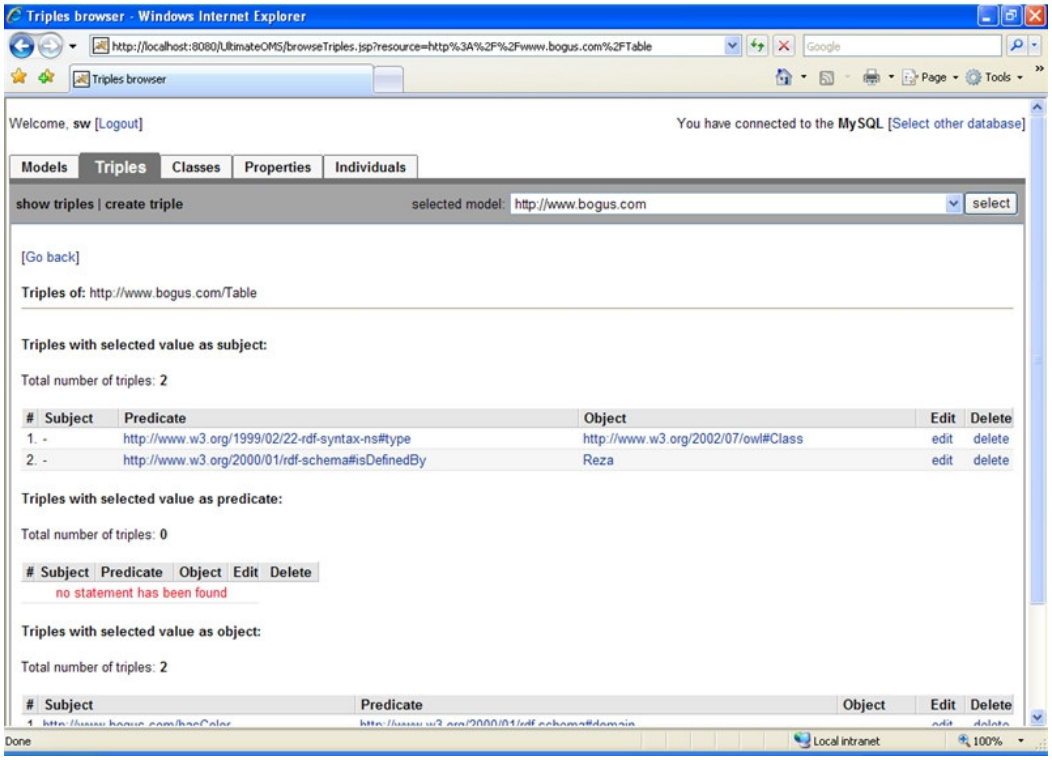

Figure C.11: Triple browsing page in UltimateOMS

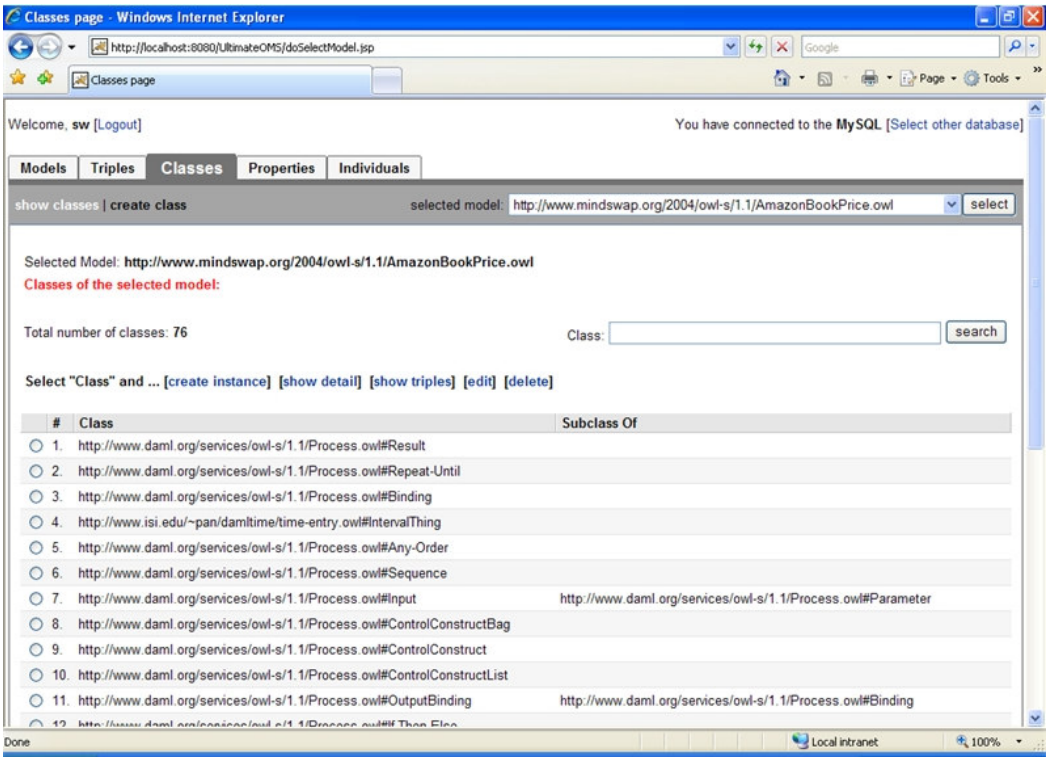

Figure C.12: UltimateOMS Classes page

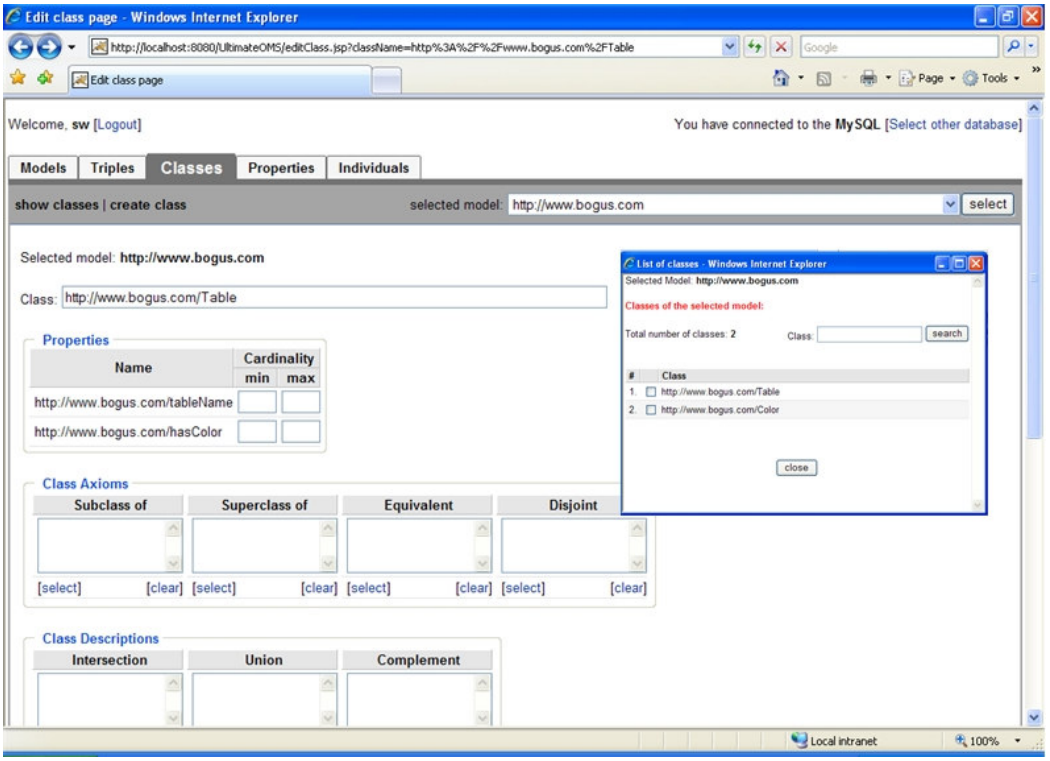

Figure C.13: Editing class page in UltimateOMS

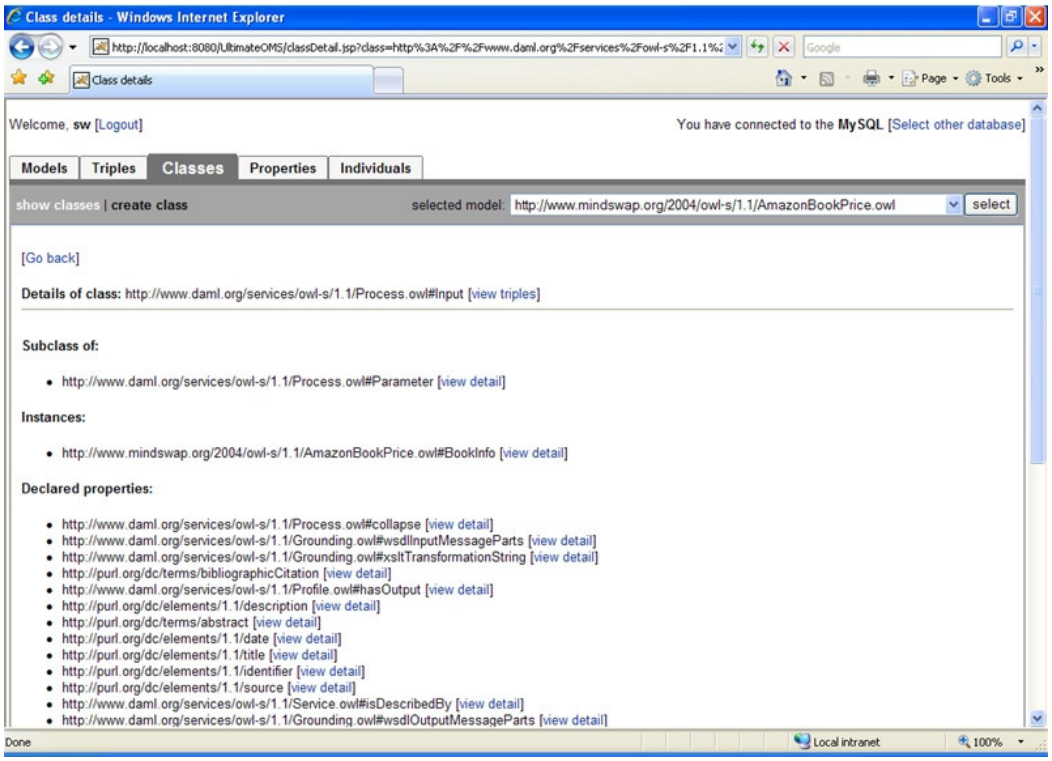

Figure C.14: Class detail page in UltimateOMS

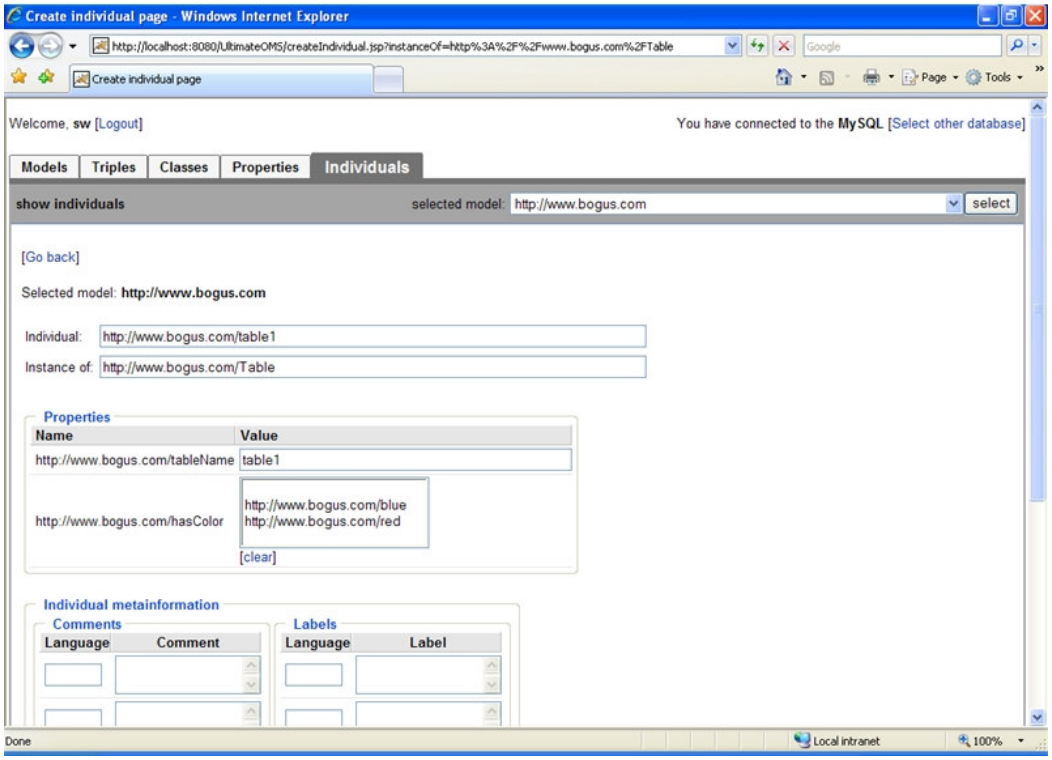

Figure C.15: Creating class instance page in UltimateOMS

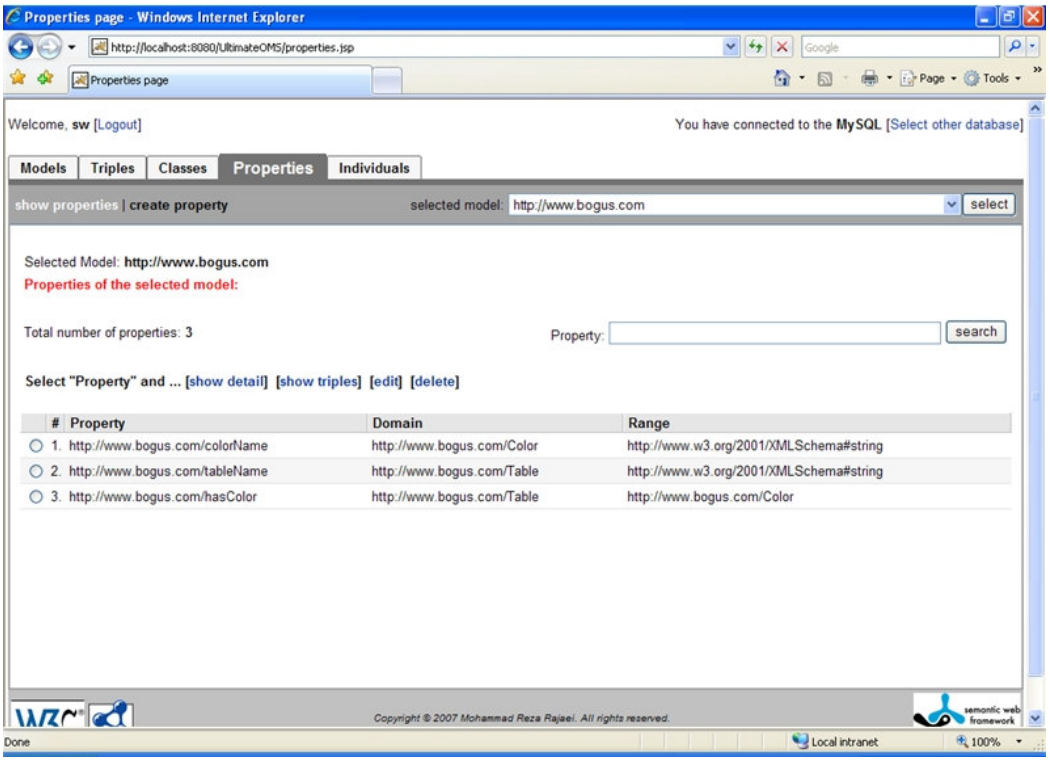

Figure C.16: UltimateOMS Properties page

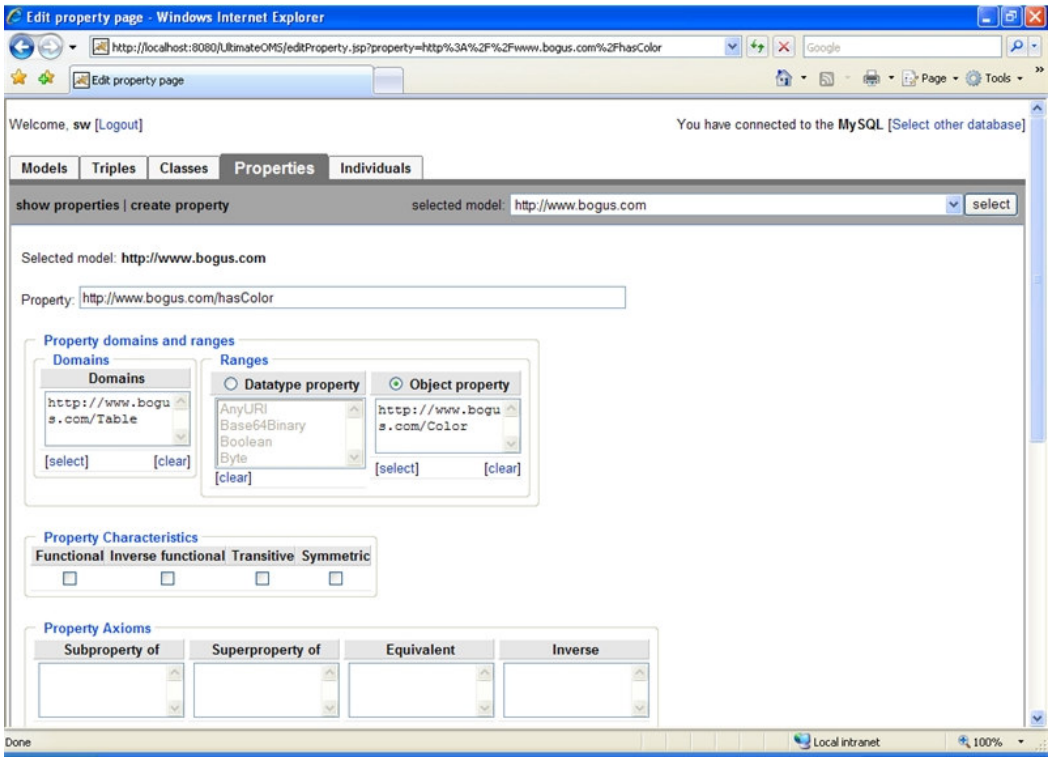

Figure C.17: Editing property page in UltimateOMS

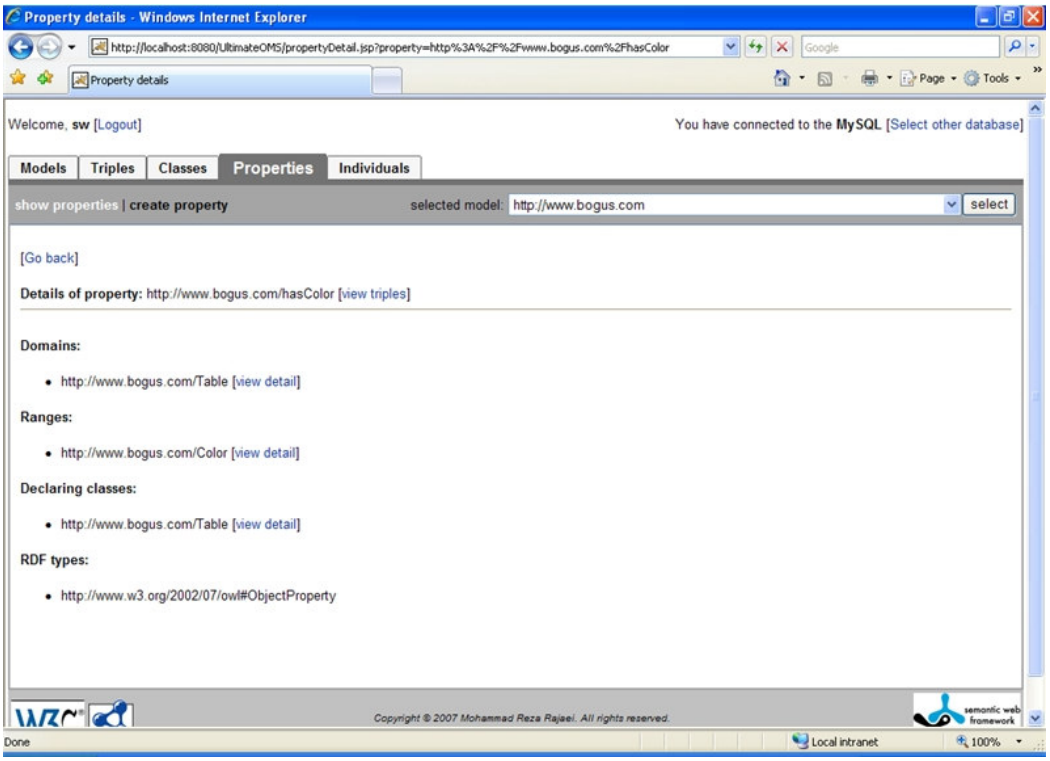

Figure C.18: Property detail page in UltimateOMS

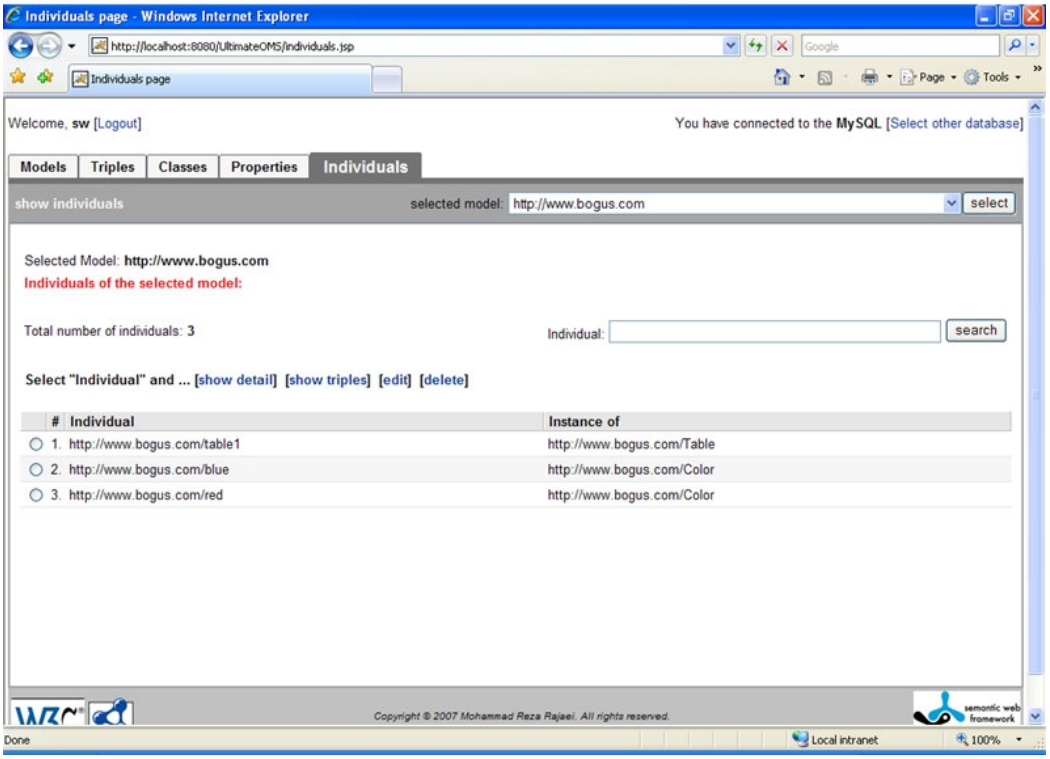

Figure C.19: UltimateOMS Individuals page

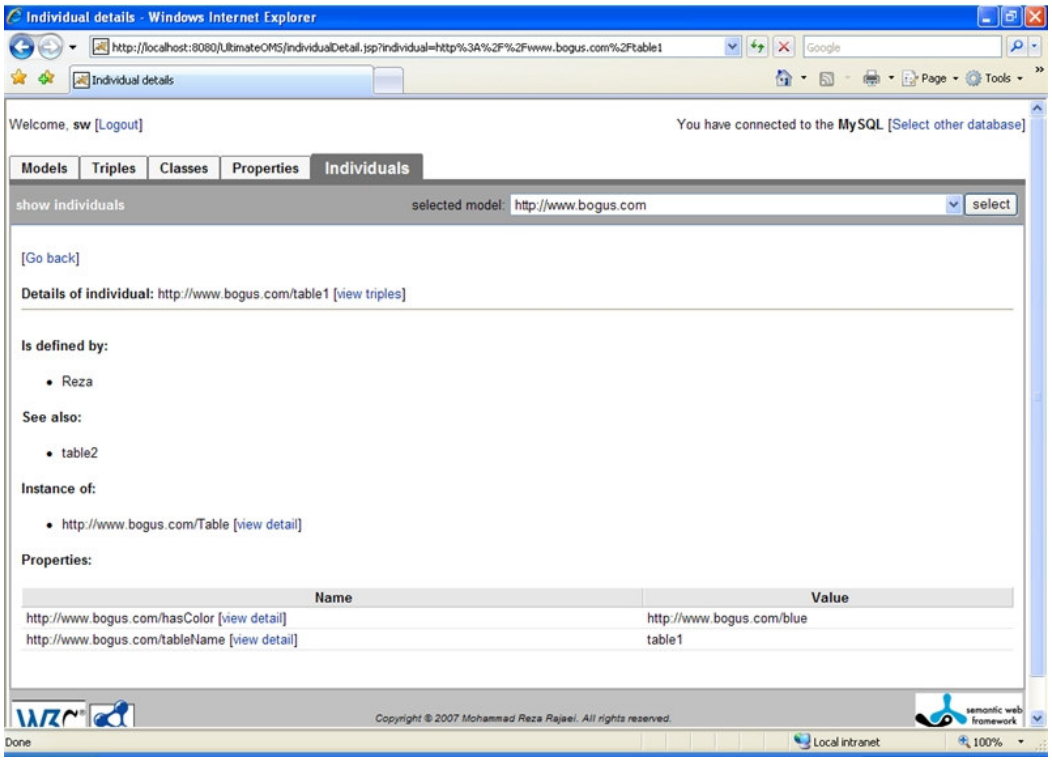

Figure C.20: Individual detail page in UltimateOMS