MASTER OF SCIENCE IN

COMPUTER SCIENCE

Vrije Universiteit Brussel

Belgium

The Creation of Adaptive Semantic Websites

September 2006

Promotor:

Prof. Geert-Jan Houben

Master dissertation in partial fulfilment of the requirements for the Degree of **Master of Science in Computer Science**

By: **Lamia Ali Abo Zaid**

Acknowledgements

My special thanks and gratitude to my promoter Prof. Dr. Ir. Geert-Jan Houben for his continues guidance and support during my work. Many thanks for his continuous discussions and encouragement while preparing this work. Many thanks for his efforts in reviewing and correcting this thesis manuscript.

I would like to thank Dr. Sven Casteleyn for his comments during this research.

I would like to express my thanks for Dr. Jeen Broekstra for his support with Sesame.

I would like to express my special thanks to my parents for their encouragement and support they gave me during my study.

I would like to thank my husband so much for his support and effort, and for bearing me during the times while I was very busy.

Last but not least, many thanks to all people who may I forgot to mention their names.

Abstract

The Semantic Web has brought the way to machine processable data by giving data *meaning* or *semantics*. Although Semantic Web is still in its infancy, it promises the development of applications that have better understanding for the data and can easily communicate and exchange knowledge. The Semantic Web can be considered one big knowledge base where all the applications would share and exchange knowledge.

User-adaptive websites and systems are gaining more popularity in these last years due to the massive amount of information on the internet. Adaptation aims at providing the users with what they need, in other words *tailor* information presented based on the user requirements.

This thesis studies the creation of adaptive Semantic Websites. We have studied the combination of both Semantic Web and the creation of User-adaptive content. We argue that semantic web brings more semantics to knowledge which will enhance the ability to create user-adaptive systems. Our main research goal was to explore how rich information provided by the semantic web can be combined together and used to satisfy the user needs. Creating a user-adaptive system we only present to the user the portion of information that meets his preferences, the source of this information is the distributed environment of the Semantic Web. We have also done a case study to give an example of how semantic web adaptation would work.

Keywords: **Semantic Web, Semantics, adaptation , user-adaptive, personalization, knowledge base, ontology.**

Table of Contents

List of Figures

List of Tables

List of Listings

Chapter 1: Introduction

1.1. Semantic Web Vision

After more than two decades since the invention of the World Wide Web, Tim Berners-Lee proposes the Semantic Web. The Semantic Web simply means the web of meaning. In the World Wide Web information is presented in natural human language which is not rich enough to convey formal meaning and therefore it is not machine processable. This current web contains millions and millions of resources such as HTML files, documents, images and graphics, and media files. These resources contain huge amounts of information scattered in various web pages and documents. The current web is a web of documents and understandable only to humans. This makes information retrieval processes very hard; humans alone cannot deal with this huge amount of resources on the web. Software agents or machines could help in this process but a difficulty arises from the fact that machines do not understand human language. Trying to make machines act as humans is a very complex task and needs a lot of training. The idea of the Semantic Web was introduced mainly to solve the problem that content on the current web is intended only for human consumption. The basic idea of the Semantic Web is to give information a well-defined meaning, thus better enabling agents and people to work in cooperation [1]. W3C states [2] ***"The Semantic Web is about two things. It is about common formats for interchange of data, where on the original Web we only had interchange of documents. Also it is about language for recording how the data relates to real world objects. That allows a person, or a machine, to start off in one database, and then move through an unending set of databases which are connected not by wires but by being about the same thing"***. We can simply say that the Semantic Web is a web of data rather than a web of documents. Semantic Web is about two things: It is about common formats for interchange of data, as opposed to documents. This data is well-defined so that agents will fully comprehend the semantics of the data. Also it is about language for recording how the data relates to real world objects. That allows a person, or a machine, to start off in one knowledge base, and then move through an unending set of knowledge bases which are linked by being about the same or related domains. We can think of the Semantic Web as a mesh of information linked up in such a way as to be easily processable by machines, on a global scale. We can think of it as being an efficient way of representing data on the World Wide Web, or as a globally linked database. The challenge of the Semantic Web was to provide a language that expresses both data and rules for reasoning about the data and that allows rules from any existing knowledge-representation system to be exported onto the Web. As stated by Berners-Lee [1] ***"Making the language for the rules as expressive as needed to allow the Web to reason as widely as desired"***.

The Semantic Web uses RDF (Resources Description Framework) to represent information. Each piece of information on the Semantic Web is called a *resource* and each resource is uniquely identified. Information about resources is represented as a Directed Graph of triples *(Subject, Predicate, Object)* also called RDF statements. Knowledge on the Semantic Web is stored in an *ontology*. The ontology holds both the data and metadata, which enables understanding the semantics. The Semantic Web structure enables not only combining Semantic Web statements to create larger pieces of information but also the ability to infer new information based on the rules defined in it's ontology. Figure 1 shows the different layers of the Semantic Web. The three upper layers are still under construction and are not final yet. Logic or reasoning is one of the major important issues for Semantic Web and it is an important design issue when creating a Semantic Web agent. The amount of reasoning an agent should do without increasing its complexity and execution cost is a big challenge. The Semantic Web structure will be further discussed in chapter 2.

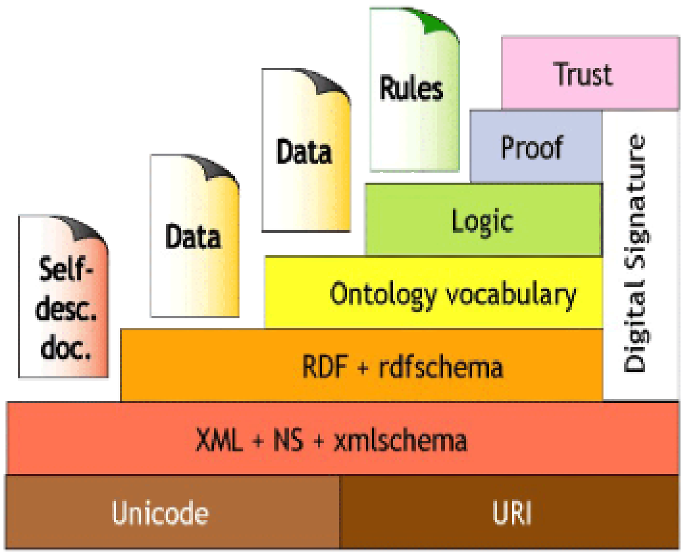


Figure 1 Semantic Web layers [5]

1.2. Moving from World Wide Web to Semantic Web

Semantic Web is not intended to replace the existing web but rather it is an extension of the current Web. The current web depends on visual representation of information through HTML tags. This visual representation makes information clear for humans to understand but very difficult for machines to understand and process. For example to emphasize something it could be in a different font or colour. Some form of extraction is required to strip off the information part from the presentation part. Other techniques are used to infer *meaning* from this information; this leads to an increased complexity in the agents dealing with the World Wide Web. Another problem with the current web is the fact that different terms are used to represent the same meaning, for example in a shopping site could refer to the shopping cart as cart, while another would refer to it as shopping basket or basket for short, yet another site could refer to it as shopping bag. All these words refer to the same meaning or the same *semantic*s, which is very obvious to humans while it is unknown to software agents. These agents have to be explicitly informed that the previous terms are all the same. Another example comes from the fact that the web is multi lingual; an English shopping website would use the word *price* to refer to an items price, while a Dutch website would use the word *prijs*, a French website would use the word *prix*, a Spanish site would use the word *precio*, an Italian site would use the word *prezzo*, and an Arabic site would use the word *الثمن*. An agent that is looking for a product and comparing prices to retrieve a list of the cheapest sites would have to be familiar with these terms. These are just a sample of languages that exist on the web while there are many more. The Semantic Web targets solving these problems by providing not only the data but also metadata that describes explicitly what this data means. This form of data annotation makes an agent understand the semantics behind the data and thus allows for better interpretation between data and gents and allows for better inter agent communication and collaboration. As stated by Berners-Lee [4], "***this notion of being able to semantically link various resources (documents, images, people, concepts, etc) is an important one. With this we can begin to move from the current Web of simple hyperlinks to a more expressive semantically rich Web, a Web where we can incrementally add meaning and express a whole new set of relationships (hasLocation, worksFor, isAuthorOf, hasSubjectOf, dependsOn, etc) among resources, making explicit the particular contextual relationships that are implicit in the current Web. This will open new doors for effective***

***information integration, management and automated services*".** The Semantic Web promises a solution in which the web becomes one big knowledge base and everyone has access to it. In order for this to happen there should be supporting technology that allows for such annotation in a formal and unified syntax, such annotation are RDF/RDFS and OWL which are standards set by the W3C. Also reasoning on the Semantic Web promises for more intelligence in services provided by the web such as personalized notifying agents, search agents, personalized search agents, e-learning and many other applications where agents would pull the information and process it having a better understanding of its meaning.

1.3. Semantic Web Personalization. Explains in fact the need for adaptive Semantic Website

Adaptive (user-adaptive) websites have been a great success over the internet in the past years. Sites like Amazon which provide a service that take into accounts the user interests is very helpful to its users and facilitates the site's use. What personalized sites do is that they personalize contents and structure according to the need of the users but in the end the contents come from their database. So it is only dynamic in the since in that it takes the user preferences into account when displaying the contents and links in their pages. A more interesting scenario would be a user that wants to buy a particular item such as a house, and uses an agent to search for all possible buys. The user would give the agent an initial feed of the specifications of the house required. This agent would search all the available knowledge bases for such an item and retrieve a set of matching houses available. However, it would also use its knowledge of the data and reasoning support to provide in this result set houses that are *relevant enough* to the user, for example houses that are near by the zone defined by the user or with a slightly higher price than the one stated by the user. It would display these matches and the user can accept or reject the matches done by the agent. The agent would record this so that the next time it runs the search it will take these accept/reject actions by the user into account and will tune its actions based on the user preferences. We can also assume that the agent will run periodically and will send new relevant matches to the user. The agent can also be adaptive to the user by checking for the user status: if he is online then it will send him the results in a mail messages or else it will send it to his mobile or PDA according to the user preferences. This scenario will be made possible with the use of Semantic Web infrastructure for the creation of these adaptive websites. First because the data annotation provided by Semantic Web provides a better understanding of the semantics behind the data, better enabling agents to do their job and obtain relevant matches that better suit the user needs. Second reasoning support provided by the Semantic Web will enable agents to create links between data enabling better understanding of user needs and providing richer solutions. The process of creating personalized websites will be more efficient with the capabilities of the Semantic Web, enabling for a better implementation of the personalization process through Semantic Web Agents. A personal agent on the Semantic Web will receive some tasks and preferences from the person, seek information from Web sources, communicate with other agents, compare information about user requirements and preferences, select certain choices, and give answers to the user [6].

1.4. Proposed work

In this thesis we tackle the problem of Semantic Web adaptation and personalization. In section 1.3 we claimed that Semantic Web provides a rich platform for creating adaptive websites. Semantic Web enables adaptation of the knowledge retrieval process. Semantic Web can be looked at as a global knowledge base. Our key motivation was to show how Semantic Web could provide knowledge that is adaptable to the user needs and preferences. We propose a general adaptive Semantic Web system structure, which can be used in any application domain. During our research we attempted to answer the following questions:

1. To which extent does Semantic Web provide rich knowledge representation?
2. How can we combine more than one ontology to create a bigger world? How will that effect the composition of queries?
3. Can we generate rich queries which provide knowledge tailored to the user needs?
4. How can the user model be taken into account in the query generation process?

1.5. Overview

Chapter 2, of this thesis introduces the Semantic Web technology to the reader. First we started by giving a detailed overview of the layers of the Semantic Web, then we discussed the Semantic

Web languages. These languages impose different complexity levels for knowledge representation. We then introduced the term *ontology* and discussed knowledge representation via ontology. Finally we introduced Semantic Web search, as a demonstration of a Semantic Web application.

In chapter 3, we discussed the problem of Semantic Web personalization. We argued that Semantic Web provides an infrastructure for rich knowledge representation, which enables the retrieval of knowledge via creating Semantic Web queries. These queries make use of the semantics within the knowledge to retrieve personalized results. In this chapter we also proposed a model for the Semantic Web adaptation process taking the user model into account.

In chapter 4, we gave an overview of existing Semantic Web tools. We discussed in detail both Protégé and Sesame, which we used in our case study.

In chapter 5, we implemented a case study to demonstrate the ability of rich knowledge representation on the Semantic Web. We also explored creating semantically rich queries for the adaptive retrieval of knowledge. This chapter demonstrates how adaptation can be achieved.

In chapter 6, we give our research results and recommendations for future work.

Chapter 2: Semantic Websites

2.1. Introduction

The Semantic Web deals with data as resources. Each resource identifies a certain piece of information, for example: a name of a person, an email of a person, a job of a person, the price of an item, and time of a show. Each resource has a description which tells what this resource is about. Resources can be linked to other resources, and this linkage is what forms the Semantic Web mesh. Thus the Semantic Web needs a different architecture than WWW in order to achieve its goals. Semantic Web has the architecture shown in figure 2 which is also called the ***layer cake*** architecture. Each layer gives more capabilities and power to the Semantic Web than the layer below it. For the minimum requirements of building the Semantic Web at least the first three layers have to be used. Figure 2 also shows that the Semantic Web is built on top of existing web architectures namely URI and XML.

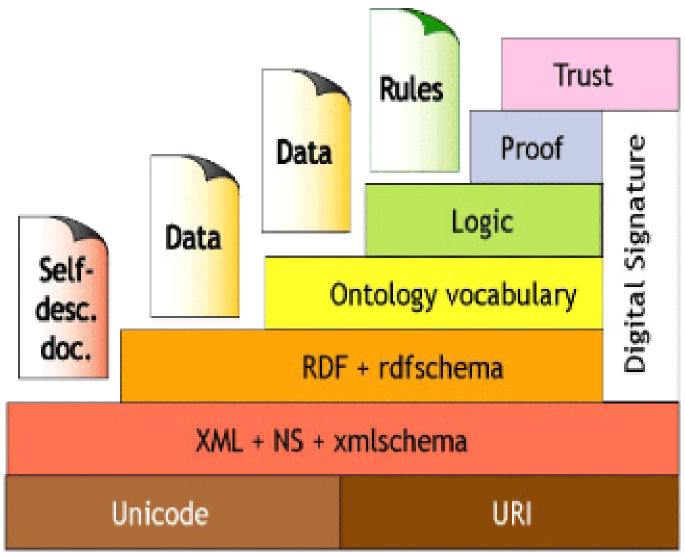


Figure 2 Semantic Web layers [5]

The Semantic Web architecture consists of the following concepts/layers:

1. **URI**

A URI (Uniform Resource Identifier) is a Web identifier: like the strings starting with "http:" or "ftp:" often found on the World Wide Web. A resource can be anything that has identity and an identifier is an object that can act as a reference to something that has identity. Anyone can create a URI, and the ownership of it is clearly delegated, so they form an ideal base technology with which to build the Semantic Web on top of. All URIs share the property that different persons or organizations can independently create them, and use them to identify resources that are either on the web or used in their organizational data model on the Semantic Web. They are built on top of Unicode which means that they support all languages and not just English. A URI reference (or URIref) is a URI, together with an optional fragment identifier at the end. For example, the URI reference http://www.vub.ac.be/students/lamia#email consists of the URI http://www.vub.ac.be/students/lamia, and the fragment identifier email, separated by the "#" character [6,7].

2. XML & XML Namespace

XML provides a robust and durable format for information storage and transmission. That is why it is used for serialization of RDF and RDFS; it provides a common syntax for the exchange of information between applications and platforms. One important feature about XML serialization of RDF or RDFS models is that it is not unique. This is due to the fact that RDF and RDFS correspond to graphs while XML is a tree structure representation. In the meantime all possible serializations of an RDF graph are valid ones. A first advantage we gain from using XML to serialize RDF and RDFS is the use of existing XML parsers. Second advantage is that XSLT can be applied to add presentation to the data serialized. Third, RDF can use values represented according to XML schema data types, thus assisting the exchange of information between RDF and other XML applications [3, 8]. XML namespaces are also used in RDF/XML serialization. A namespace is a collection of element and attribute names identified by a Uniform Resource Identifier reference. The reference may appear in the root element as a value of the ***xmlns*** attribute. The RDF namespace URI reference (or namespace name) is http://www.w3.org/1999/02/22-rdf-syntax-ns# and is typically used in XML with the prefix rdf although other prefix strings may be used. The RDF Vocabulary is identified by this namespace name. More than one namespace may appear in a single XML document, to allow a name to be used more than once. Each reference can declare a prefix to be used by each name [9]. Using namespaces in RDF/XML serialization makes it easier to read for humans.

XML has no predefined tags; each organization puts the tags that better describes their data. When an XML document conforms to a certain schema we can be sure that this XML document is using the elements defined within the schema [10]. For this reason the W3C has put an RDF schema [11] which defines the terms used to describe information in RDF. This creates a standardized form of describing information in RDF.

3. Semantic Web Languages

The three layers above the XML layer represent the Semantic Web languages. These are the languages used for creating Semantic Web content, the higher the layer the more complex the language capabilities become. These languages do not only provide a way to represent data but also they represent knowledge. One important aspect about knowledge representation is that it must provide clear unambiguous mapping of real world objects. Knowledge must be represented in a formal way in order to remove any ambiguity in interpreting this knowledge whether by humans or machines. Semantic Web languages fall into three main categories based on their complexity, namely:

- Describing Web resources and their relations; such as: RDF, RDFS
- Modeling contexts, domains and interpretations these are called Ontology Languages; such as: DAML[1], OWL
- Annotating web resources with rules and policies (RuleML[2], SWRL[3])

4. Reasoning

Knowledge representation is a set of assertions and assertions about assertions which should not be contradictory. The basic model contains just the concept of an assertion, and the concept of quotation [12]. Putting assertions together we could infer new assertions, this is called reasoning. The logic layer in Semantic Web povides the reasoning capabilities. There exists a number of Logic reasoners for web ontology languages such as Fact[4], Racer[5] and Pellet[6].

[1] http://www.daml.org/

[2] http://www.ruleml.org/

[3] http://www.w3.org/Submission/SWRL/

[4] http://www.cs.man.ac.uk/~horrocks/FaCT/

[5] http://www.sts.tu-harburg.de/~r.f.moeller/racer/

[6] http://www.mindswap.org/2003/pellet/

5. **Proof and Trust**

These layers are still under construction. If an agent would commit to certain ontology then that ontology must be correct. That means the terms in the ontology are correct, the restrictions in the ontology are correct and the relations that could be inferred from this ontology are also correct and are true. One suggestion for how this process should be is stated by Berners -Lee: ***"The proof will be a chain of assertions and reasoning rules with pointers to all the supporting material"*** [12]. Another possible scenario is that: ***"The Logic layer enables the writing of rules while the Proof layer executes the rules and evaluates together with the Trust layer mechanism for applications whether to trust the given proof or not"*** [13]. Digital Signature can be used to populate trust on the Semantic Web. A reasoner would have to validate the signature of the source of the information or rules from where it got proof to a certain fact, if it is a trusted signature it will continue and save these facts as true or else it would report un-trusted data has been found and the action to take would depend on the level of trustworthiness required.

Semantic Web layers provide the specifications and structure that would guide Semantic Web tools and thus make Semantic Web work at its full potential. Semantic Web standardization is another important feature that will allow the Semantic Web to become one big knowledge base. W3C has set RDF/RDFS and OWL as the standard Semantic Web languages. We will discuss them in more details in the coming sections.

2.2. Semantic Content in RDF(S)

The Semantic Web uses RDF (Resources Description Framework) to represent information. The basic building block of information on the Semantic Web is called a Resource and each resource is uniquely identified by giving it a unique identifier or URI. RDF is the base for representing any knowledge on the Semantic Web; it is considered a Metadata language for representing resources. An RDF triple is the smallest unit used for data representation on the Semantic Web. Information about resources is represented as a Directed Graph (DG) of (Subject,Predicate,Object) triples also called RDF statements. Subject is always a resource, object could be a resource or a string and predicate (also called property) is the relation between the subject and object which is also a resource. Figure 3 shows an RDF statement. The nodes of an RDF graph are the set of subjects and objects of triples in the graph, while the arcs represent the relation between the subject and object. This is equivalent to the conventional method in representing knowledge in terms of relations in FOL (First Order Logic) format.

An RDF statement would be represented as Predicate (Subject , Object) in FOL. Resources are linked together to create a mesh of related triples on the Semantic Web. An RDF graph is the main representation for RDF statements.

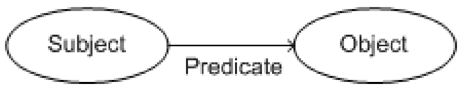


Figure 3: Graphical representation of an RDF Tripe

RDF has formal semantics which provides a dependable basis for reasoning about the meaning of an RDF expression but yet it has no restriction on the vocabulary used for describing the resources. In RDF we can formalize any statements about any resources using the appropriate domain vocabularies. One thing to keep in mind is that in RDF we represent a binary relation between subject and object i.e. P(S,O), while in FOL a relation could hold between any number of inputs i.e. P(t_1, $t_{2,}$ $t_{3, \ldots,}$ t_n). This requires some form of decomposition technique to allow for RDF representation of such relations. The simplest form of decomposition would be to say that the subject is a Blank Node [3] and thus we have the following triples: (Blank Node, P, t_1) , (Blank Node, P, t_2) , (Blank Node, P, t_3) …..(Blank Node, P, t_n).

Example 1: Figure 4 shows the statement *Lamia is a student in VUB in Brussels* represented in RDF. This is a mapping for the FOL relation: *Student (University, VUB, Brussels)*

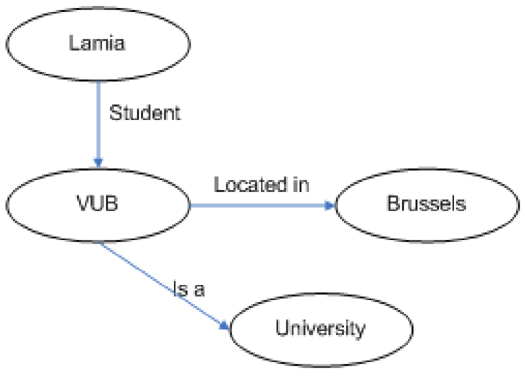


Figure 4: Example RDF representation

2.2.1. RDF Representations

There exist a number of notations to represent an RDF graph model, some of them are:

- **RDF/XML serialization:** Provides an XML serialization for RDF graphs. An RDF graph can be considered a collection of paths of the form node-arc-node. In order to encode the RDF graph in XML, the nodes and predicates have to be represented in XML terms, such as element names, attribute names, element contents, and attribute values. Therefore, in RDF/XML these turn into sequences of elements inside elements which alternate between elements for nodes and predicate arcs. This has been called a series of node/arc stripes. The node at the start of the sequence turns into the outermost element, the next predicate arc turns into a child element, and so on. The stripes generally start at the top of an RDF/XML document and always begin with nodes. [6]. The RDF/XML serialization for the statement *lamia is a student in VUB* which would be represented by the following triple *(lamia, student, vub)* is shown in listing 1.

```
<?xml version="1.0"?>
<rdf:RDF     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:std="http://vub.ac.be/students#"
xmlns:univ="http://vub.ac.be/university#" >
      <rdf:Description rdf:about="http://vub.ac.be/students#lamia">
            <std:student>
                  <rdf:Description
rdf:about="http://vub.ac.be/university#vub"/>
            </std:student>
      </rdf:Description>
</rdf:RDF>
```

Listing 1: RDF/XML representation for (lamia, student, vub)

- **N-Triple:** is a line-based, plain text format for representing RDF. Each triple must be written on a separate line and has a dot in the end [15]. The N-triple notation for the triple *(lamia, student, vub)* is shown in listing 2

```
<std:lamia> <http://vub.ac.be/students#student> <univ:vub> .
```

Listing 2: N-Triple representation for (lamia, student, vub)

- **Notation3 (N3):** is a shorthand non-XML serialization of RDF, designed with human-readability in mind. A dot indicates the end of the RDF triple. Namespaces are defined by the @prefix keyword. Notation 3 allows for making shortcuts when having several statements about the same subject: a semicolon (;) introduces another property of the same subject, and a comma (,) introduces another object with the same predicate and subject. [14]. The N3 notation for the triple *(lamia, student, vub)* is shown in listing 3.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix std: <http://vub.ac.be/students#> .
@prefix univ: <http://vub.ac.be/university#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

# Data statements
<std:lamia> std:student <univ:vub> .
```

Listing 3: N3 representation for (lamia, student, vub)

2.2.2. RDF Schema (RDFS)

RDF is a metadata language that does not provide special vocabulary for describing the resources. It is often essential to be able to describe more of a subject than saying it is a resource. Some form of classification for these resources is often required to be able to be able to provide a more precise and correct mapping of the world. The basic idea behind Semantic Web is to provide meaning of resources, as defined in the Knowledge Representation domain, ***"knowledge is descriptive and can be expressed in a declarative form"*** [16]. The formalization of knowledge in declarative form begins with a **conceptualization**. This formalization includes the objects presumed or hypothesized to exist in the world. This is why RDF schema (RDFS) was introduced as a language that provides formal conceptualization of the world. RDF Schema semantically extends RDF to enable us to talk about classes of resources, and the properties that will be used with them. The RDF schema defines the terms that will be used in RDF statements and gives specific meanings to them. It provides mechanisms for describing groups of related resources and the relationships between these resources. Meaning in RDF is expressed through reference to the schema. RDFS consists of a collection of RDF resources that can be used to describe properties of other RDF resources this makes it a simple ontology language which allows more capture of semantics than just pure RDF. The most important resources described in RDFS are:

- **Classes:** RDFS deals with classes through the term ***rdfs:Class*** which defines a class. RDFS allows for creating hierarchies of classes in which a class is defined as the subclass of another class using the ***rdfs:subClassOf*** property. RDFS also allows for creating instances of a class; that is data that are of the type of that class. The ***rdf:type*** property may be used to state that a resource is an instance of a class. RDFS defines a special class ***rdfs:Literal*** which is the class of literal values such as strings and integers, ***rdfs:Litera***l is an instance of ***rdfs:Class*** [11].

- **Properties:** A property (also known as predicate) is the relation between a subject resource and an object resource, it is defined by ***rdf:Property***. RDFS allows also for hierarchies of properties defined by the ***rdfs:subPropertyOf*** property which states one property is a subproperty of another. Two other properties associated with the RDF property are: ***rdfs:range*** and ***rdfs:domain*** which allow for restrictions of the property values. The triple *(P, rdfs:range, C)* states that P is an instance of the class ***rdf:Property***, that C is an instance of the class ***rdfs:Class*** and that the resources denoted by the objects of triples whose predicate is P are instances of the class C. While ***rdfs:domain*** is instance of rdfs:Property that is used to indicate the class that will have as members any resource that has the indicated property. The triple *(C, rdfs:domain, P)* states that the class C has as its member any class having the indicated property P [11].

- **RDF terms:** RDFS semantically extends RDF therefore it defines RDF terms such as: ***rdf:Statement***, ***rdf:subject***, ***rdf:predicate***, ***rdf:object***

Referring again to example 1, we can represent it in RDFS as shown in figure 5. This new representation is richer in semantics than the one of figure 4. First some new classes have been defined; *Student* which has an instance *Lamia* and *City* which has an instance *Brussels*. Some new properties have been defined; *studies in* which has a domain *Student* and a range *University* and *Located in* which has a domain *University* and range *City*.

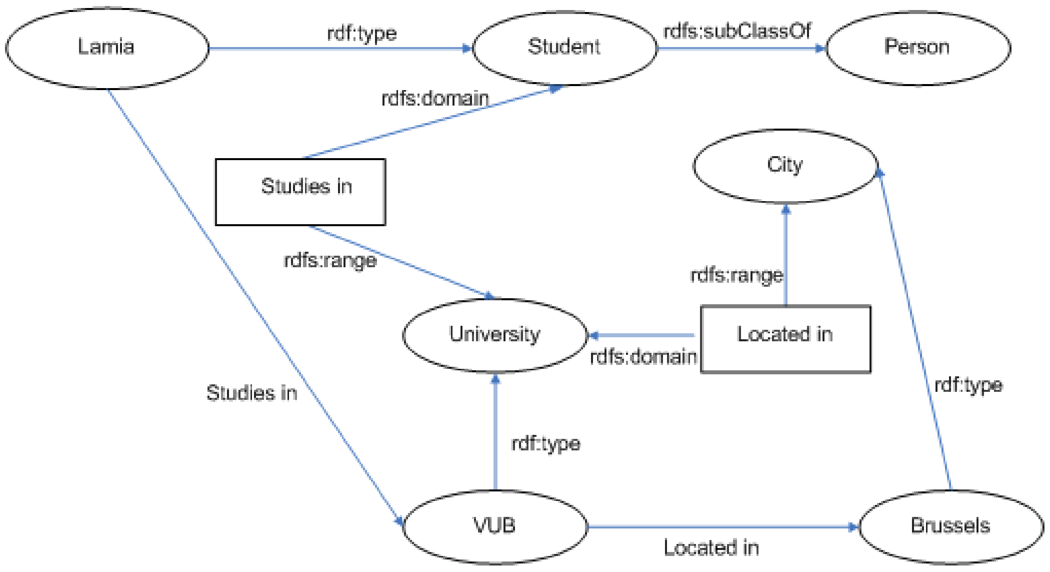


Figure 5: Example RDF representation

It must be noted that the richer the ontology language becomes the more expressive it allows representing term and thus agents committed to this ontology gain more semantics out of the knowledge expressed.

2.3. Semantic Content in OWL

OWL (Web Ontology Language) was introduced to provide richer vocabulary than RDFS. OWL semantically extends RDF/RDFS which means that an OWL ontology is an RDF graph. A formal semantics describes the meaning of knowledge precisely and rich semantics describes fine grained knowledge. OWL was introduced by W3C to provide a richer ontology language that allows for: a well-defined syntax, efficient reasoning support, a formal semantics and sufficient expressive power [7]. The main content of OWL ontology is carried in its axioms and facts, which provide information about classes, properties, and individuals in the ontology. There are a several major capabilities that OWL adds to RDF and RDFS. The first is the ability to create local range restrictions. In RDFS, a property is allowed to have only one class as its range while in many cases there is a need defining more restrictions on the property range. An example would be having an *eats* property in a *Person* class and we want to specify that *Person eats Food* so we add a range restriction that the value for *eats* must come from a class *Food*. This is true for people in general. However, if we want to express vegetarian persons then we would have a subclass of *Person* called *Vegetarian*. We also want to specify that vegetarians eat a subset of food which is *Vegetarian Food.* However, using only RDF and RDFS, there is no way to say that people, in general, eat food while vegetarians do not eat meat. In OWL, we can leave the domain and range on the *eats* property unchanged, and state in the *Vegetarian* class that vegetarians are restricted to eating only vegetarian food; i.e. the value for the *eats* property must come from the class *Vegetarian Food*. These types of restrictions on property ranges allow authors to create more expressive ontologies and for agents to make more logical assumptions. For example, if we know a person is a vegetarian, then we know anything they eat must be an instance of the *Vegetarian Food* class [17]. OWL also introduces basic set functionality, such as unions, intersections, complements, and disjointness. As an example if we have two subclasses of *Person* called *Male*, *Female*, we would want to restrict individuals to be an instance of either *Male* or *Female* but not both. In RDF/RDFs there is nothing to prohibit defining an individual as both *Male* or *Female* while in OWL we could identify that these two classes are disjoint, this will prohibit to define individuals belonging to both classes. OWL introduces many other new capabilities of an ontology language but differing according to which species of OWL you are using. OWL has three species: OWL Lite, OWL DL, OWL Full they have the following specifications:

- **OWL Lite:** OWL Lite was designed for easy implementation and to provide users with a functional subset that will get them started in the use of OWL. It is a subset of OWL DL and has a lower formal complexity than OWL DL [19].
- **OWL DL:** OWL DL, where DL stands for ***Description Logic***, was designed to support the existing Description Logic business segment and to provide a language subset that has desirable computational properties for reasoning systems. It provides maximum expressiveness while retaining computational completeness, i.e. all conclusions are guaranteed to be computable; and decidability, i.e. all computations will finish in finite time [19][20].

- **OWL Full:** OWL full is the complete OWL language, it relaxes some of the constraints on OWL DL so as to make available features which may be of use to many database and knowledge representation systems, but which violate the constraints of Description Logic reasoners. [19]

2.3.1. Axioms and Facts in OWL

Axioms are used to associate class and property identifiers with either partial or complete specifications of their characteristics, and to give other information about classes and properties. Facts state information about a particular individual, in the form of classes that the individual belongs to, and also properties and values of that individual [18].

- **OWL Class Axioms**: OWL deals with classes through the term ***owl:Class*** which is a subclass of ***rdfs:Class***. Additionly all RDFS class terms are valid for use in OWL. The OWL owl:sameClassAs defines that two classes are the same and thus have the same individuals. OWL allows for boolean combination of classes via: ***owl:intersectionOf***, owl:unionOf, owl:complementOf. OWL also allows for some restrictions on classes for example ***owl:disjointWith*** allows to define a class as disjoint with another, this guarantees that an individual belonging to both classes will not be allowed by the OWL reasoner. OWL supports class value constraints through ***owl:allValuesFrom*** which defines possible ranges for a certain property in a class. The owl:someValuesFrom constraint is analogous to the existential quantifier of FOL, it states that for each instance of the class that is being defined there exists at least one value for P that fulfills the constraint.

- **OWL Property Axioms:** In addition to the RDFS property terms OWL adds ***owl:ObjectProperty*** and ***owl:DatatypeProperty***. The ***owl:ObjectProperty*** is used to identify relations between instances of two classes. The ***owl:DatatypeProperty*** is used to identify relations between instances of classes and RDF literals and XML Schema datatypes. Defining the domain, range and subproperty of a property are still valid and done using the RDFS terms. OWL allows adding relations between properties such as: ***owl:inverseOf*** and ***owl:equivalentProperty.*** The term ***owl:inverseOf*** defines an inverse relation between two properties, normally properties have a direction from domain to range ***owl:inverseOf*** make a bidirectional relation. The term ***owl:equivalentProperty*** defines two properties to be the same value.

- **OWL Facts:** OWL defines some facts such as ***owl:sameAs*** which links an individual to an individual stating they are the same individual. ***owl:differentFrom*** states that two URI references refer to different individuals. ***owl:AllDifferent*** provides a way for stating that a list of individuals are all different, this means they are pairwise disjoint.

2.3.2. OWL Ontology Sharing and Reuse Support

One important feature of OWL is its support for sharing and reusing ontologies. An ontology once defined can be used by importing it by other ontologies through the ***owl:imports*** statement which references it to another OWL ontology containing definitions, whose meaning is considered to be part of the meaning of the importing ontology. Each reference consists of a URI specifying from where the ontology is to be imported. The ***owl:imports*** statements are transitive, that is, if ontology A imports B, and B imports C, then A imports both B and C [19]. The need for importing ontologies will be discussed further in the next section.

2.4. Using Ontologies in Semantic Websites

Semantic Websites depend on semantic contents which are represented in RDF/RDFS or OWL. Figure 6 shows a sample of how Semantic Web documents might obtain their contents. The Semantic Website will probably provide some service, like with conventional web documents the data is obtained from a database; only in Semantic Web it is called a knowledge base. A knowledge base is an ontology containing both classes and their properties and individual instances of these classes. As with conventional databases query languages such as SQL enable retrieval of data (records) from the database, also in knowledge bases query languages will retrieve of data (triples) that make a match. But there is more to it, knowledge base and ontology are not just another database, but rather they provide *semantics* and *data*.

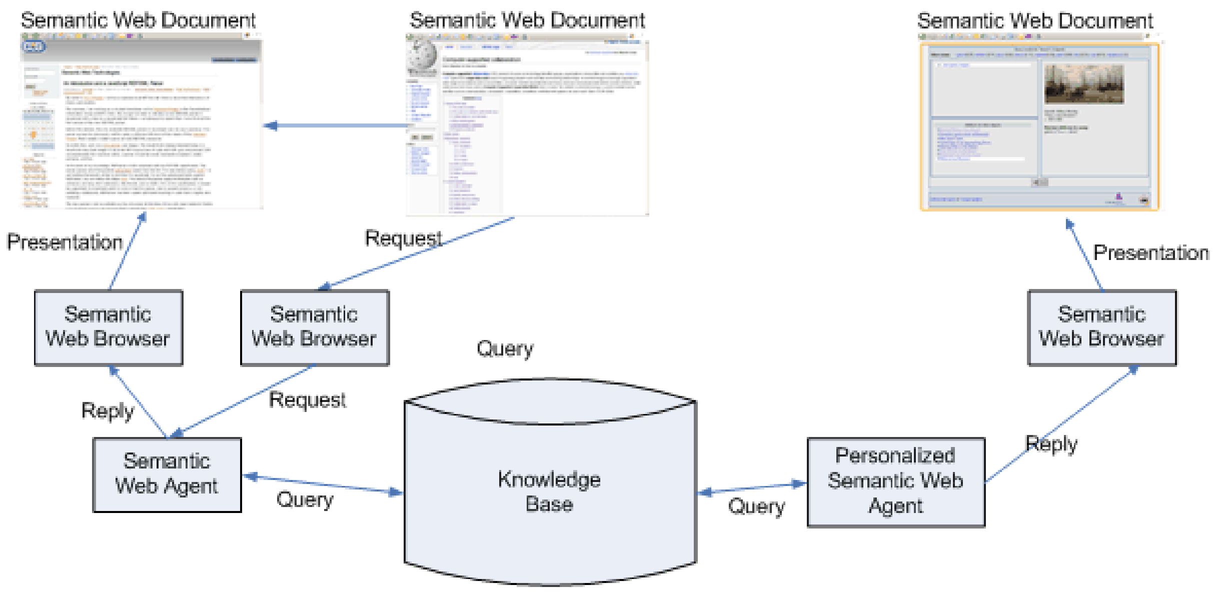


Figure 6: Semantic Web Documents, a sample architecture

2.4.1. What is an Ontology?

Ontology is a term shared from philosophy, it means the study of being or existence. Ontology is that branch of philosophy which deals with the nature and the organization of reality. Artificial intelligence borrowed this term from philosophy to represent a shared well agreed upon conceptualization. There are a number of definitions for ontology in literature some of them are:

- An Ontology is a specification of a conceptualization [22]

- Ontologies are agreements about shared conceptualizations. Shared conceptualizations include conceptual frameworks for modelling domain knowledge; content-specific protocols for communication among inter-operating agents; and agreements about the representation of particular domain theories. In the knowledge sharing context, ontologies are specified in the form of definitions of representational vocabulary.[23]

- An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of phenomena in the world by having identified the relevant concepts of those phenomena. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine readable. Shared reflects that ontology should capture consensual knowledge accepted by the communities.[24]

- An Ontology is a set of explicit shared formal concepts which permits Categorization. It is used to organize or classify content (or information, or knowledge). The concepts have names (words) and definitions (meanings). They are used to talk about the content. They are metadata [25].

From the above definitions we can conclude that an ontology represents a shared conceptualization of a certain domain. Domain objects, their properties and relations between these objects as well as constraints that may exist on either the objects or their properties are represented in the ontology. These representations are well-defined and thus understandable and are made available for all agents to use. The part of the world conceptualized described is called the knowledge domain. The ontology is a knowledge representation.

2.4.2. Importance of Ontologies

Even before Semantic Web there is a rapid growth in information volume and the need for sharing this volume. There is a need for sharing information on the web, between organizations through intranets, among researchers, between internet agents and distributed collaborating softwares of some organizations. This need calls for a standardization of terms and concepts. Many disciplines now develop standardized ontologies that domain experts can use to share and annotate information in their fields. This allows for [20] [23] [24]:

- Ontologies facilitate knowledge sharing and reuse between agents by providing a consensual and formal conceptualization of a given domain. This allows for sharing a common understanding of the structure of information and enhances information exchanges between organizations and on the internet.
- Separating the domain knowledge from the operational knowledge and allow making explicit domain assumptions which can easily be modified or maintained later.

- The domain model implicit in an ontology can be taken as a unifying structure for giving information a common representation and semantics.

Once an Ontology is modelled and constructed agents start using it to interpret knowledge based on how it is defined in the ontology. Agents that use terms of a certain ontology are said to be ***committed*** to that ontology. An agent is said to ***commit*** to an ontology if its observable actions are consistent with the definitions in that ontology. As stated by Gruber: ***"a conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly. An Ontology is that explicit specification of a conceptualization"*** [23]. When agents commit to a certain ontology that means they use and understand the semantics of the objects, properties and constraints within that ontology. All agents committed to the same ontology can share knowledge in between them in a consistent and easy manner as knowledge becomes understandable due to the shared ontology.

2.4.3. Knowledge Representation in Ontologies

An ontology expresses knowledge or *the world* in terms of *classes*, *properties* and *restrictions*. Classes represent the real world concepts or objects. Similar to their existence in real world classes can form a taxonomy of classes which all belong to the same inheritance hierarchy, or have an IS-A relationship. A concept in real world has features and attributes, these are mapped into an ontology as properties of a class. Finally there exist some relations between the classes and their properties or between classes and each other. These relations are called *role restrictions* and they map as restrictions in the ontology. We say that an ontology is an interpretation of the world that focuses on data and their relations, which should reflect the semantics of this data forming a broader view than just describing data such as in databases. Ontologies can also hold individuals which are instances of the classes described in the ontology, in this case an ontology becomes an knowledge base.

The vision of the Semantic Web was to produce a web that is consumable by agents as well as humans. It provides data with well-defined meaning removing any ambiguity or misinterpretation. This allows for easy agent collaboration, opening an opportunity for the creation of Semantic Web agents with many services to provide. Semantic Web also leaves no chance for misunderstanding of data or confusion of terms since an agent would interpret the meaning based on the ontology it is committed to. Thus ontologies form the backbone of the Semantic Web. Agents committed to the same ontology can easily share information because the terms are understandable. This makes ontology sharing a very important concept for the Semantic Web success: the more popular the ontology becomes the more benefit is gained from using it. What would really enrich Semantic Web is the development of standard ontologies for researchers and users to start using. We think the Semantic Web will really work with its full potential when there exists ontologies that feature all problem domains and thus it would be a matter of selecting the correct ontology to commit your agents from. Some projects of building large ontologies for a specific domain have already been launched and some of them already ready and set to work some of these ontologies are: Gene Ontology [7], Ontology for Biological Data [8], and Ontology for Geographic Information - Spatial Schema [9]. We could characterize the users on the Semantic Web to: ontology builders who are responsible for modelling and creating an ontology for a certain domain, Semantic Web developers using the existing ontologies to create Semantic Websites and end users who will deal with Semantic Web via Semantic Web agents and Semantic Web browsers. To make the process of finding the right ontology to use easier, there exists a number of ontology repositories that provide many ontologies featuring many domains. Some of the most famous of such repositories are: DAML repository[10], Stanford OWL repository[11], and SchemaWeb[12].

2.5. Combining Ontologies to create a larger world

In real world problems often there exist terminologies from more than one domain. Figure 7 shows that in real world applications there is a connection to more than one domain which means

[7] http://www.geneontology.org/

[8] http://www.biopax.org/

[9] http://loki.cae.drexel.edu/~wbs/ontology/iso-19107.htm

[10] http://www.daml.org/ontologies/

[11] http://protege.stanford.edu/plugins/owl/owl-library/

[12] http://www.schemaweb.info/

that the application would probably link to more than one ontology. From each ontology it will use only the terms that verify its needs based on the problem it targets.

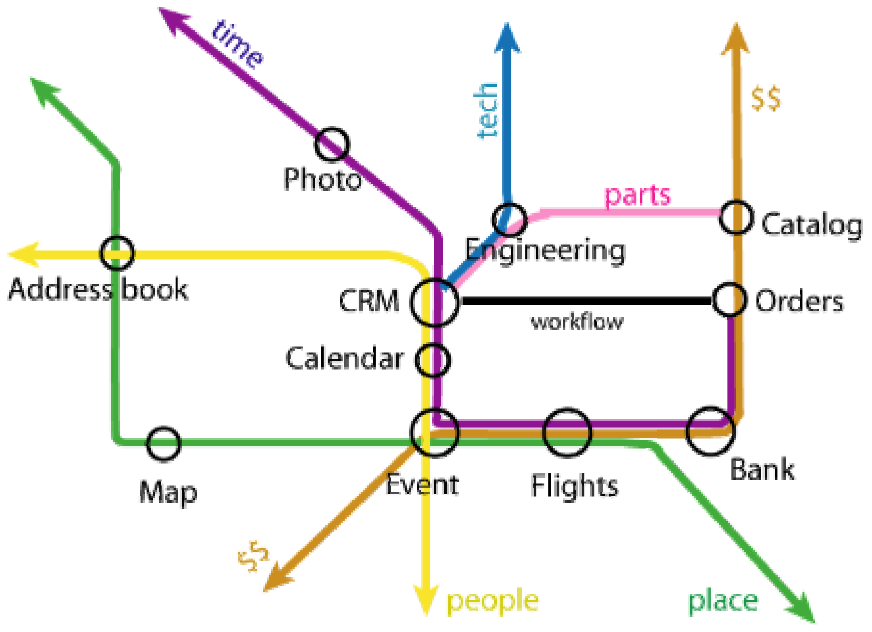


Figure 7: Applications are connected by common concepts [26]

It must be noted that ontologies are general and are not tailored to a specific problem rather they are tied to a specific domain: an ontology maps all the concepts, axioms and facts of that certain domain. Larger more general ontologies are created by combing terms of existing ontologies, this is called ontology reuse. An important aspect of ontology reuse over Semantic Web is to create an ontology using the terms and structure in another ontology, thus combing one or more ontologies to create a larger one. A knowledge base can have individuals based on one or more ontologies. Having only one standard language for ontology editing will greatly influence the sharing and reusing, that is why OWL was introduced by W3C. As mentioned before OWL facilitates this by providing the owl:imports term to indicate importing an ontology. Figure 8 shows an example of such collaboration on the Semantic Web, where agents would commit to an ontology or knowledge base and also allows for agents committed to the same ontology or sharing the same terms to collaborate based on a common shared understanding of terms. Figure 8 also shows that ontologies can share the terms existing in one ontology and also can reuse these terms by building new terms taking the old ones as their base. In this case the use of namespaces is very important to avoid name clashes and to enable defining new terms in terms of existing ones.

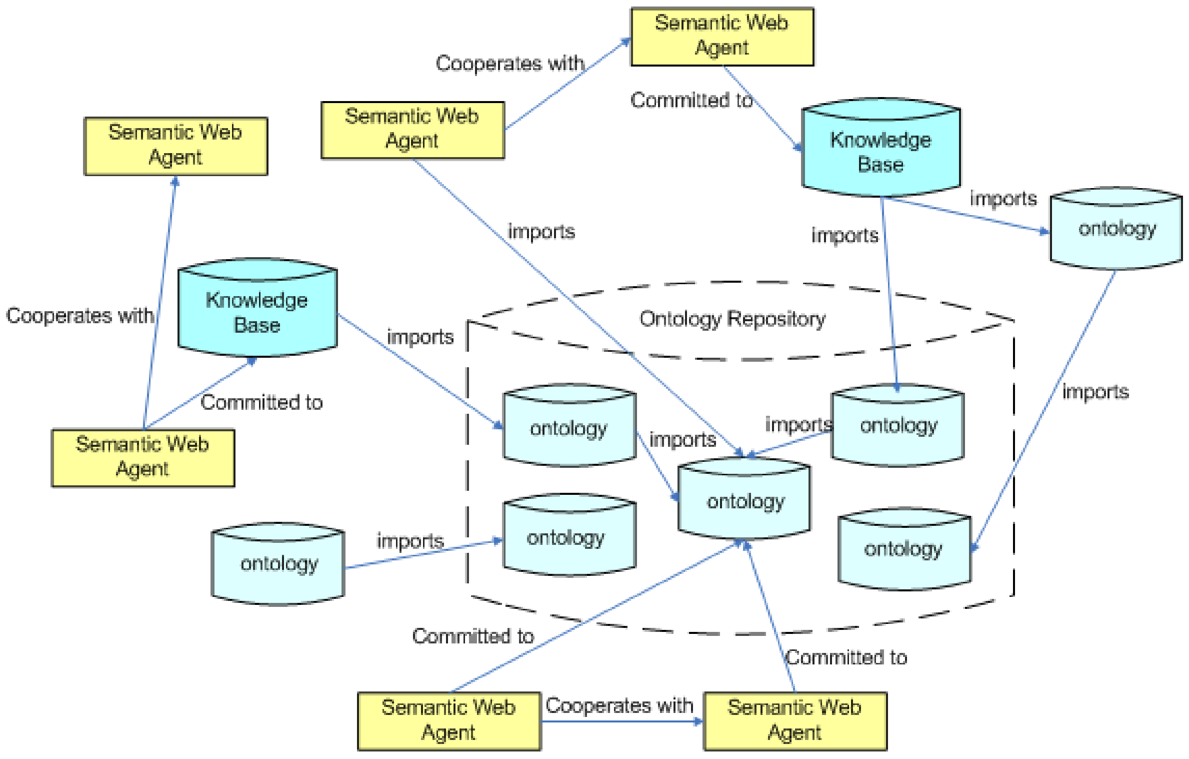


Figure 8: Example of agent collaboration and ontology commitment scenario on the Semantic Web

2.6. Searching the Semantic Websites

As the Semantic Web grows in size there will be an increasing need of searching for information. Users will want to perform search queries and expect Semantic Web documents that best match their query as results, in analogy to WWW search; only the expected precision of Semantic Web search should be better due to the better understanding of the search terms. It must be noted that Semantic Web documents are different than conventional web documents, information viewed in Semantic Websites are what the developer wants to present but the semantics behind the presentation is what matters, while with web documents the presentation s simply the way of formatting the looks of the information. For example a computer shopping site in Dutch, Arabic, and English would share the terms from a computer shopping ontology, while the terms are presented in the three different languages but their semantics is the same because they have a common source of semantics. A search engine searching for certain computer specifications would perform its search and return the result based on the user preferred presentation. In this section we will show the analogy between Semantic Web search and WWW search, the requirements of Semantic Web search and the expectations, and finally give some examples of current semantic search agents. Let us first discuss WWW search:

2.6.1. Information Retrieval on WWW: WWW search

The World Wide Web contains many documents that are meant to be human understandable. Information is presented in natural language text, which is hard for agents to understand, as they cannot infer meaning. Search engines on the web evaluate their search based on one of these methods:

- **Web Directories or indexing:** they manually organize web pages into a taxonomy of topics. Indexing is done for main keywords in a webpage. Yahoo directories used this technique.
- **Web crawler**: agents called crawlers visit the web sites, read the information on the actual site, read the site's meta tags and also follow the links that the site connects to. Webcrawler[13] uses this technique.
- **Web crawling and Web indexing**: agents called crawlers visit a web site, read the information on the actual site, read the site's meta tags and also follow the links that the site connects to performing indexing on all linked web sites as well. The crawler returns all that information back to a central repository, where the data is indexed. Google uses this technique.

In all of these methods search is done based on string matching with some lexical analysis done to make the searching process more intelligent [28]. For example searching for OWL, the search engine cannot know whether you mean OWL the bird or the ontology language. While writing the query as *"OWL"+"semantic web"* will retrieve all the documents related to the Semantic

[13] http://www.webcrawler.com/

Web. The search results depend a lot on the creativity of the user in writing the search query. The results are the set of documents that match the search criteria or provide a good approximation for it.

The order in which the search results are returned depends on ranking criteria. Some search engines use frequency and the positioning of keywords to determine relevancy, others use page popularity [28] [29]. Ranking is one of the important issues in search engine design as it greatly affects the user satisfaction.

2.6.2. Semantic Web Search

The Semantic Web currently holds many ontologies and knowledge bases or what is called Semantic Web documents [30]. An ontology whether written in RDF/RDFS or OWL; is a document with significant proportions of its statements defining new terms such as new classes and properties, or extends the definition of terms defined in other ontologies. A Knowledge base defines individuals and makes assertions about them, or makes assertions about individuals defined in other knowledge bases. A search facility is required so that Semantic Web users can query existing Semantic Web documents or search for their existence. Users could also query for certain triples that could be inferred from more than one document. Semantic Web search is the process of knowledge retrieval on the Semantic Web. In the previous section we discussed WWW search. These WWW methods are not appropriate for Semantic Web search for the following reasons:

1. Semantic Web is data oriented; it represents a mesh of related data, while the WWW is a mesh of related documents.

2. Semantic Web was introduced to be machine understandable; data exists along with its well-defined meaning which will help agents in the information retrieval process.

3. WWW documents are text-based documents while Semantic Web documents can be a mixture of concrete facts, classes and property definitions, logic constraints and metadata, even within a single document. There is a need for a matching of concepts and not a matching of strings in semantic search queries.

4. Some reasoning capabilities are required for a Semantic Web agent to deduct additional facts, constraints and relations that may exist between the data. There may be a need to combine data from more than one source to fully understand its semantics and infer new facts or constraints.

5. Semantic Web data is neither text-based nor document-based. Which implies that a new ranking criteria should be found which are more suitable for the Semantic Web documents.

6. It must be noticed that Semantic Web documents have many file extensions such as *.rdf*, *.rdfs*, *.owl* and *.xml*. The file format could be any valid RDF representation such as N Triple, N3, and RDF/XML. This means that the search engine used would be able to understand these formats and do linkage between all valid Semantic Web documents related to the same topic.

7. There is a trend to embed RDF content in XHTML documents. Also Adobe has adopted RDF as an extensible way to embed metadata in images, PDF documents and other data formats [31].

In spite of their differences, the interface of the Semantic Web search engine should resemble traditional keyword based web search engines. Queries should be accepted and results generated based on the search algorithm used. Search Queries would be submitted in plaintext. Text may be quoted to treat multiple words as a single unit and use of and/or logical combinations should also be acceptable. There should be an option to search for Triples or existing documents.

2.6.3. Searching the Semantic Web: what to look for?

Information retrieval on the Semantic Web can be addressed according to three different points of view: developers of ontologies, focusing on the representation of domain knowledge, annotators of web resources, creating semantic annotations based on ontologies, and end-users**,** asking ontology-based queries for searching the web [31]. In this case the requirements of a Semantic Web search engine would be:

1. Find an ontology containing specified terms based on user search query, users should be able to qualify the type of the term whether it is a class or a property.
2. Find Instances based on user search query, and based on the integration of data from many knowledge bases.
3. Find portions of metadata and data based on user search query and semantic reasoning capabilities of the search engine.
4. Capability to restrict the search to a certain type of ontology.

In order to achieve its goals the Semantic Web engine needs to do these steps:

1. Discover and index Semantic Web documents.
2. Discover and index inter-data relations based on applying reasoning to visited Semantic Web documents.
3. Process user queries
4. Order search results

1. **Discover and index Semantic Web documents**

Similar to WWW search engines which have their own directories, Semantic Web search engines also can index semantic documents. RDF databases can be used to store this data. The point worth noting is scalability; the required database is expected to hold millions of entries and will soon grow more and more as the Semantic Web grows. RDF repositories can be used for this purpose.

Discovering Semantic Web documents means crawling the web for all available Semantic Web documents. A Semantic Web crawler or a scutter does this job. Like in WWW search engines a typical crawler starts from a set of seed URLs, visits documents, and traverses the web by following the hyperlinks found in visited documents. The fact that the Web forms a well-connected graph and the ability for people to manually submit new URLs make this an effective process. A Semantic Web

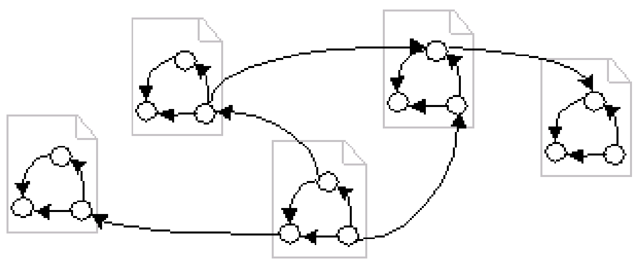


Figure 9: Semantic Web crawling, moving from node to node [27]

crawler must deal with several problems. Semantic Web documents are not very dense on the web and are not yet all linked and well connected to one another, due to the fact that Semantic Web is in its start. This yields that an exhaustive crawl of the web is not an efficient approach. As shown in figure 9 in a Semantic Web document a node could yield to crawling to another one in a separate document and as the number of nodes increases this would become very branching. Many of the URLs found in a Semantic Web document point to documents, which are not Semantic Web documents. Following these can be computationally expensive, so heuristics to limit and prune candidate links are beneficial [31]. It will be good practice to start crawling from seeds that come from the index database and start crawling all the available Semantic Web documents within the same directory or sub directories. Also parsing the Semantic Web document for import statements enables crawling all the imported ontologies and/or knowledge bases. Other things to search for are [30]:

1. Namespace of a URIref that links from a resource reference to a resource definition.

2. URLs of the instances of owl:Ontology, owl:imports.

3. URL of resource in special triples, such as triples using rdfs:seeAlso, rdfs:subPropertyOf, rdfs:subclassOf, and owl:equivalentClass.

4. Other URIs will need a filtering approach so the crawler can determine whether or not to crawl them. If they represent resources such as documents, images or other. The crawler will not crawl them. If they represents another Semantic Web

document or contains a namespace of another Semantic Web document then they will be crawled.

New discovered Semantic Web documents should be added to the Semantic Web engine database and documents, which have been modified, should be updated.

2. **Discover and index inter-data relations**

A single Semantic Web document may not reveal all the knowledge about a certain domain. Information could be scattered in more than one document but linked. This comes from RDF capability to describe any resource, which is uniquely defined by its URI. Combing together these documents to have a wider overview is an expensive task but yet it represents one of the key features of Semantic Web, which is the ability to reason on the data. One way to reduce this cost is by doing reasoning only when it is required (i.e. based on the search query). The crawler should have reasoning capabilities and be able to link terms that would be significant for the search without prior knowledge of the domain.

Figure 10 shows a simple portion of an RDF graph. From this graph there is straightforward information such as *Lamia is a Student*, and inferred information for example, the fact that a student is a subclass of person then we could infer that *Lamia is a person*. Also from the facts: *"Lamia is a member in Wise Lab"*, "Wise Lab part of VUB" this should lead to *"lamia is a member in VUB",* with adding a constraint that a member is a student if it has type student, then Lamia would be a student in VUB. This reasoning capability is important as it provides more knowledge, which arises from linking the facts. It can also be used to provide the nearest hits and not just exact hits. For example if the user is searching for *Lamia's* homepage, instead of no results, the search engine could provide the nearest result in the RDF hierarchy, in this example it would return *http://wise.vub.ac.be* which is the homepage of the object where Lamia is a member. Defining how far to go in the hierarchy is an important design issue. The more far we go the more insignificant the results would be and yet a certain level of approximation is required. The crawler must be able to extract this information from any valid representation of Semantic Web metadata/ data.

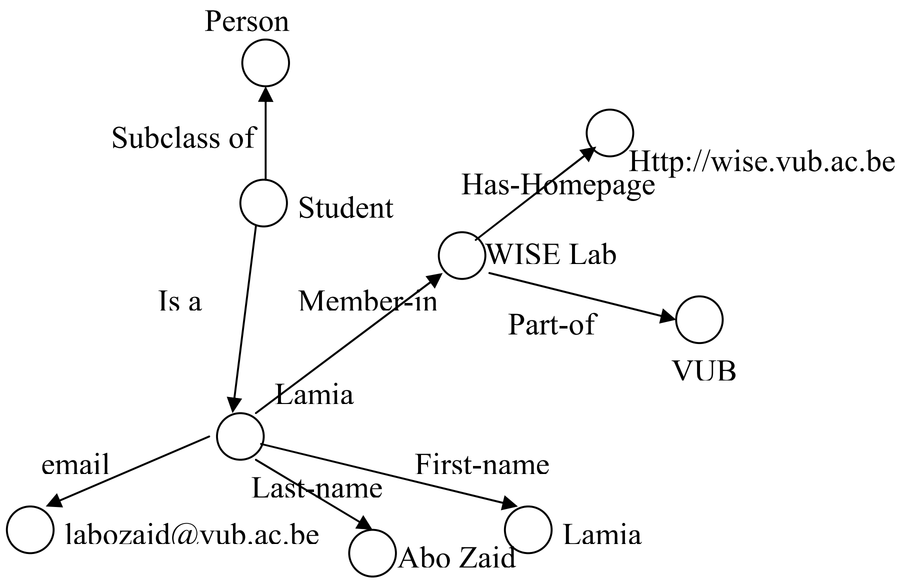


Figure 10: A portion of an RDF graph

3. Process user queries

User queries could include constraints on the content metadata and relation metadata or it could be a free text string. If the user query is a text search string then it is possible to find expert rules that are not explicitly stated in the rule base, but which can be constructed by regular combination of existing rules. Retrieved rules can be ranked semantically, according to the distance in the hierarchy between the term, which is searched for and the term, which is retrieved. In case that no results are found, we could change our search, searching for terms upper in the hierarchy. This is a type of *over-generalisation*. For example, a query that searches for *'Students'* can be replaced by query for *'all People'*, [32]. It should be noted that how far in the hierarchy the engine should go, is one of its design issues.

For handling queries that contain content metadata or relation metadata we must note that, the Semantic Web data can be aggregated at several levels of granularity. Figure 11 shows these different levels of granularity. It ranges from the universal graph of all RDF data on the web to a single RDF triple and the term URIs it comprises.

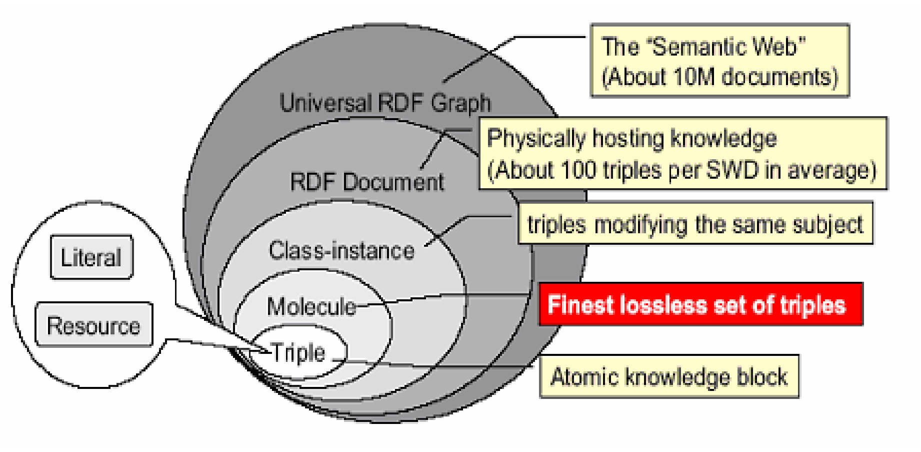

The "Semantic Web" (About 10M documents)
Universal RDF Graph
Physically hosting knowledge (About 100 triples per SWD in average)
RDF Document
triples modifying the same subject
Class-instance
Literal
Resource
Molecule
Finest lossless set of triples
Triple
Atomic knowledge block

Figure 11: levels of granularity for Semantic Web data [31]

Since search engines usually return the references or location of search results, we could identify three types of output at different levels [31].

1. **The term URI:** At the lowest level is a URI representing a single RDF term – a class, property or instance. For Semantic Web content, these terms are analogous to words in natural language. Knowing the appropriate terms used to describe a domain is an essential requirement for constructing Semantic Web queries.

2. **An RDF Graph**: In order to access knowledge in the Semantic Web, users would need to fetch an arbitrary sub-graph from a target RDF graph. The sub-graph might correspond to a named graph, a collection of triples with a common subject, or an RDF molecule.

3. **The URL of a Semantic Web Document:** This corresponds to the result returned by a conventional web search engine – a reference to the physical document that serializes an RDF graph.

User queries should be passed to a query processor, which analyzes existing terms and constraints used. It then constructs a valid Semantic Web query and queries the database as shown in figure 12. If the user query was free text then the query would be a match with all available string resources in the database, this match should be done the same way as in web search engines.

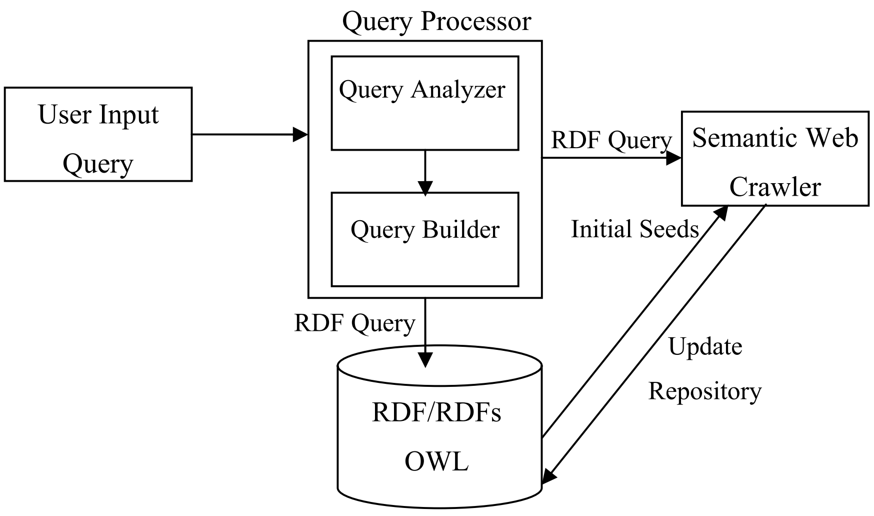


Figure 12: Semantic Web search Engine- query processing

4. Order search results

One of the important features of a search engine is how well this engine sorts its results; this is referred to as ranking algorithm. Efficient ranking is one of the reasons why Google is such a success. Results should be sorted based on closeness of the match. If A and B are Semantic Web documents that are results of some query, we propose some features that the ranking algorithm could do:

1. A comes before B if both A and B have a hit at the same level (no generalization used) and A imports B or A uses B.
2. A comes before B if A has a hit at a lower level than B (A less generalized than B).
3. A come before B if A has the search term occurred more than B.
4. In case of a complex search query where a number of terms are used, weighted sum mechanism can be used to calculate which is more relevant A or B.

2.6.4. Current Semantic Web Search engines

In this section we will explore existing Semantic Web engines. Currently there is no fixed model for how a Semantic Web engine should be. Currently the most popular search engine doe Semantic Web is Swoogle, which we have described in some detail.

1. **Intellidimension Semantic Web Search**

   Intellidimension Semantic Web Search Engine[14] is a meta search engine. It searches for resources in Semantic Web documents and outputs the terms or resources where the search query was found. Even more, it allows users to search for a specific term that they know exists and identify how this resources appears in the triple and what value it could take. For example it allows you to specifically search for persons, from the FOAF vocabulary [33], (foaf:Person) that have a first name (foaf:firstName) of 'Eric' and a last name (foaf:surname) of 'Miller'; in this case users will construct the following search query you would enter the search string *'[foaf:surname]~Miller [foaf:firstName]~Eric'*. This search will be efficient if the user knows the structure of what he is looking for inside a certain ontology or knowledge base or at least know the vocabulary of its underlying schema.

   - **Search technique used:**
     Intellidimension Semantic Web Search Engine crawls the Semantic Web and indexes RDF documents based on the information they contain. It creates its own distributed RDF model that describes the contents and location of all documents on the Semantic Web. Since the information is described using RDF vocabularies that have well-defined meaning to computers, as well as people, search conditions can be precisely described to Semantic Web Search using these vocabularies. A Semantic Web query is constructed and evaluated on the RDF data by a query language called *RDFQL*. So Semantic Web Search translates a search condition into an index lookup into its RDF model. It returns the locations of the documents on the Semantic Web where the

[14] http://www.semanticwebsearch.com/

information described by the search condition exists [34]. It also allows users to submit their RDF documents to be crawled and indexed by their crawler. Intellidimension Semantic Web Search Engine is built on top of Intellidimension's RDF gateway, which is a platform for creating Semantic Web applications. It has its own Query language, Inference engine with reasoning capabilities and Database repository. It supports the following RDF notations: RDF/XML, N3, and NTriples

2. Corese

Corese stands for **Conceptual Resource Search Engine**. It is an RDF engine based on Conceptual Graphs (CG). It enables the processing of RDF Schema and RDF statements within the CG formalism. Corese search engine is intended to be a query based information retrieval system. It works on information stored in ontologies. Corese has its own ontology representation language, which is built upon RDFS. It is rich enough to represent ontologies provided with a concept hierarchy and a relation hierarchy. Corese takes into account subsumption links between concepts and between relations when matching a query with an annotation. Corese ontology representation language also enables to represent domain axioms, which are important to analyze when matching a query with an annotation. Annotations are represented in RDF and related to the RDF Schema representing the ontology they are built upon. The query language is also built upon RDF; for each query, an RDF graph is generated, related to the same RDF Schema as the one of the annotations to which it is to be matched. Corese does not only find exact matches but it is able to provide the user with approximate answers to a query.

- **Corese's architecture**

  Corese undergoes a number of stages to transform an RDF ontology to a Corese ontology which can then be explored and reasoned with Corese engine and queried by the Corese query language. These steps are:

1. **Represent ontological data in Corese understandable form:** The Corese engine internally works on conceptual graphs (CG). When matching a query with an annotation. Corese translates both RDF graphs and their schema to the CG model.

Through this translation, Corese takes advantage of previous work of the Knowledge Representation leading to conceptual graph based reasoning capabilities. RDF and CG have many features in common so no knowledge is lost in the translation process. The projection operation is the basis of reasoning in the CG model. A query is thus processed in the Corese engine by projecting the corresponding CG into the CGs translating the annotations. The retrieved web resources are those for which there exists a projection of the query graph into the annotation graph [35]

2. **Reasoning in Corese:** Corese integrates an inference engine based on forward chaining production rules, which implement CG rules. The rules are applied once the annotations are loaded in the system and before the query processing occurs. Which makes the annotation graphs augmented by rule conclusions before the query graph is projected on them.

3. **Processing queries:** One important feature about Corese is its capability to find approximate answers to user queries where an exact match cannot be found. This is done by applying approximation, which evaluates the semantic distance of classes or properties in the ontology hierarchies. Thus Corese does not only retrieve web resources whose annotations are specializations of the query, it also retrieves those whose annotations have a structure upon which they are just close enough in the ontology hierarchies. Corese enables to define which concepts can be approximated and which ones must be found exactly. More generally, it enables to specify conditions on the types that are acceptable and those that are not. For this purpose Corese contains comparison operators that can be associated to each query concept. Corese also has a second approximation capability by means of the standard RDF properties that reflect such as rdfs:seeAlso and rdfs:subClassOf. The effect of such properties between two classes is to shorten the actual semantic distance between these the two classes in the Corese approximate query processing.

3. Swoogle

Swoogle[15] is a metadata and search engine for online Semantic Web documents. It analyzes these documents and their constituent parts (e.g., terms and triples) and records meaningful metadata about them. Swoogle provides web-scale Semantic Web data access service, which helps human users and software systems to find relevant documents, terms and sub-graphs, via its search and navigation services. It allows users to search for a specific term within the ontology, search for ontologies and free text search. It also allows users to submit sites that contain Semantic Web documents or that are Semantic Web documents. These sites will be verified and added to Swoogle's index. Swoogle defines Semantic Web documents (SWD) as a document in a Semantic Web language that is online and accessible to web users and software agents [30]. Figure 13 shows Swoogle architecture, which is data centric and extensible. Its components work independently and interact with one another through a database. It has four major components: **SWD discovery**, **metadata creation**, **data analysis**, and **interface**:

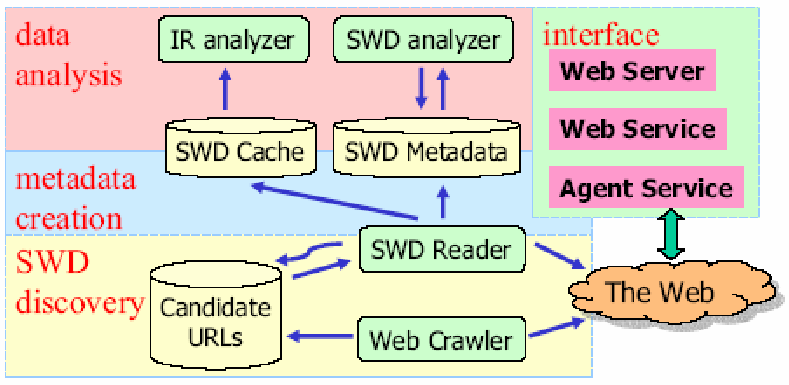


Figure 13: Swoogle architecture [30]

1. **The SWD discovery component**: discovers potential SWDs throughout the Web and keeps up-to-date information about SWDs.

2. **The metadata creation component**: caches a snapshot of a SWD and generates objective metadata about SWDs at both the syntax level and the semantic level.

[15] **http://swoogle.umbc.edu/**

3. **The data analysis component**: uses the cached SWDs and the created metadata to derive analytical reports, such as classification of SWOs (Semantic Web Ontology) and SWDBs (Semantic Web Database), rank of SWDs, and the IR index of SWDs.

4. **The interface component**: provides a web interface for the swoogle search engine where the user can enter his search query and receive the search results.

- **The SWD discovery component**

Swoogle does not use one crawler for crawling the Semantic Web instead it implements the following techniques to discover SWDs:

1. **Runs a meta-search on Google to find candidates:** Swoogle uses google's search API to provide initial seeds for the web crawler. A search is done using two of Googles criteria: the ability to do a search within a limited file type and the ability to do search within a specific domain. So first a search query is submitted to Google varying the file type to types, which could contain Semantic Web documents such as: *.rdf, rdfs, .owl, .daml,.xml* . The search results are then filtered to detect the real SWDs and a new search query is constructed, but this time with the site as a restriction. For each site where SWDs where find a query will be constructed to look for more SWDs in that site. The returned search results represent the seed for the next step.

2. **Uses a focused directory web crawler to traverse directories containing SWDs:** If a SWD was found on a website, then it is expected to find more SWDs on this same site or its subsites. This is the idea of the directory web crawler. It crawls on a certain directory to retrieve all Semantic Web documents that might exist on it, processing all web pages at HTML level. It may also find links to other sites where it would start a new crawl on these sites following the same mechanism.

3. **Invokes a custom Semantic Web crawler on discovered SWDs:** Swoogle has many Semantic Web crawlers called Swooglebot. These are different than the web

directory crawlers because Semantic Web crawlers only process SWDs. Semantic Web crawlers follow links selectively using some popular heuristics such as:

- Namespace of a URIref that links from a resource reference to a resource definition. swoogleboot has the capability to capture redirected namespaces
- URLs of the instances of owl:Ontology
- URL of resource in special triples, such as triples using rdfs:seeAlso

4. **Collects URLs of SWDs and directories containing SWDs submitted by users:** Analogous with site engine submission where site owners submit there sites to search engines. Swoogle provide a web form for collecting URLs of SWDs or URL or directories containing SWDs so that it can index it and add it to its database.

- **The metadata creation component**

  After discovering SWDs it should be possible to search for particular SWDs and Semantic Web terms. SWD metadata is collected to make SWD search more efficient and effective. It is derived from the content of SWD as well as the relation among SWDs. Swoogle identifies the following two categories of metadata:

  1. **Basic metadata:** It represents the syntactic and semantic features of a SWD. Swoogle captures the following features:
     - Encoding. It shows the syntactic encoding of a SWD. There are three existing encodings, namely "RDF/XML","N-TRIPLE" and "N3".
     - Language. It shows the language used by a SWD. Swoogle considers the usage of four meta level languages: "OWL", "DAML+OIL", "RDFS", and "RDF".
     - OWL Species. It shows the language species of a SWD written in OWL. There are three possible species: "OWL-LITE", "OWL-DL", and "OWL-FULL".

Swoogle maintains RDF statistics that summarize node distribution of the RDF graph of a SWD. In an RDF graph, a node is recognized as a class if it is not an anonymous node and it is an instance of rdfs:Class. Similarly, a node is a property if it is not an anonymous node and it is an instance of rdf:Property; an individual is a node which is an instance of any user defined class. Based on these statistics Swoogle calculates an ontology ratio, which is a number from 0 to 1. This number indicates whether the SWD is an ontology or a SWDB (knowledge base). As the number approaches 1 it is an ontology and vice versa, a threshold can be used to differentiate both, for example 0.9. [30][31]

2. **Relations among SWDs:** Along with node level processing done in step 2. Swoogle captures SWD level relations, which generalize RDF node level relations. These relations are summarized in table1.

Type	**Classes and Properties**
IM	owl:imports, daml:imports
EX	rdfs:subClassOf, rdfs:subPropertyOf, owl:disjointWith, owl:equivalentClass, owl:equivalentProperty, owl:complementOf, owl:inverseOf, owl:intersectionOf, owl:unionOf daml:sameClassAs, daml:samePropertyAs, daml:inverseOf, daml:disjoinWith, daml:complementOf, daml:unionOf daml:disjointUnionOf, daml:intersectionOf
PV	owl:priorVersion
CPV	owl:DeprecatedProperty, owl:DeprecatedClass, owl:backwardCompatibleWith
IPV	owl:incompatibleWith

Table 1: Indicators of SWD level relations

- **IM:** shows that an ontology imports another ontology. The URLs of referenced ontolgoies are collected by recording the objects in triples whose predicate is owl:imports or daml:imports.
- **EX:** shows that an ontology extends another. Such a relation may be produced by many properties.
- **PV:** shows that an ontology is a prior version of another.
- **CPV**: shows that an ontology is a prior version of and is compatible with another.
- **IPV**: shows that an ontology is a prior version of but is incompatible with another.

- **The data analysis component:**
This component uses the cached SWDs and the created metadata (created from the metadata creation component), to derive analytical reports, such as classification of SWOs and SWDBs, rank of SWDs, and the Information retrieval index of SWDs [6]. It has two main tasks:

1. **Indexing and retrieval of SWDs:** Swoogle implements traditional IR techniques to retrieve data because of their efficiency and the hardness of implementing reasoning capabilities. Based on some work related to applying IR techniques, it has been shown that traditional IR techniques can be applied to Semantic Web documents [30][36][37][38]. For the indexing mechanism, Swoogle maintains full text indexing on only the URI and literal description of the defined resource instead of the entire SWD to index the content of SWDs with better precision. For each class instance both the lexemes extracted from its URI and the literal descriptions in associated triples are good keywords.

2. **Ranking SWDs:** Swoogle uses a rational random surfing model, which accounts for the various types of links that can exist between SWDs, this is

called OntoRank algorithm. An agent either follows a link from one SWD to another with a constant probability d or jumps to a new random SWD. It is *rational* in that it jumps non-uniformly with the consideration of link semantics and models the need for agents to access the ontologies referenced in SWDs. When encountering an SWD α, the rational surfer will transitively import the official ontologies that define the classes and properties used by α. Swoogle uses the TermRank algorithm to order the RDF terms returned by a term search query. This ranks terms based on how often they are used, estimated by the number of SWDs that use the term. Swoogle uses OntoRank and TermRank, to order a collection of SWDs or RDF terms, respectively.

2.7. Reasoning and querying in Semantic Websites

Knowledge retrieval on the Semantic Web is done via querying the knowledge base for matching triples. Knowledge on the Semantic Web is distributed as it is held in multiple repositories across the Semantic Web. One advantage of representing data in formal semantics is the ability to reason on this data. By reasoning we mean the ability to infer new facts and axioms from exiting ones. The query engine must take into account any statements that are inferred from reasoning, when it is evaluated. Thus a reasoner is always part of the query engine. The reasoner will either perform the reasoning process on all the data in the knowledge base and then implement the query, or it will implement the query and start reasoning the resulted triples, in either way the result is the same [39].

A query engine should understand the semantics of RDF Schema and OWL. It should be able to do FOL based reasoning and in case of OWL support this should be extended to DL reasoning. This will add more flexibility to the query use, we don not have to query only these triples that exist explicitly but also we could query triples that are inferred from the semantics of the data. The data in Semantic Web is represented in terms of triples, this data combines to types of information, the schema information (metadata) and the actual data information. The query engine also will return as a result the set of triples that verify the query. Currently many query languages exist; of the most common is RQL[16] which is a typed functional language for querying RDF/RDFS and relies on a formal model for directed labelled graphs permitting the interpretation of superimposed resource descriptions by means of one or more RDF schemas. RQL adapts the functionality of semistructured/XML query languages to the peculiarities of RDF but, foremost, it enables to uniformly query both resource descriptions and schemas [40]. As in SQL, ***select*** specifies the number and order of retrieved data, ***from*** is used to navigate through the data model, ***where*** imposes constraints on possible solutions. For example [6], in a knowledge base where we have a *triple (staff member, phone, phone number)*, when we want to retrieve all phone numbers of staff members, we write:

```
select X,Y
from {X}phone{Y}
```

Where X will be bound to the staff member which is the subject, and Y will be bound to the phone number which is the object.

W3C has put a standard RDF query language called SPARQL. SPARQL is a query language for retrieving information from RDF graphs based on matching graph patterns [41]. The above example in SPARQL would be

```
 select ?X,?Y
where {
?X phone ?Y .
}
```

SPARQL has four query result forms. These result forms use the solutions from pattern matching to form result sets or RDF graphs. The query forms are:

- **Select**: Returns all, or a subset of, the variables bound in a query pattern match.
- **Construct**: Returns an RDF graph constructed by substituting variables in a set of triple templates.
- **Describe**: Returns an RDF graph that describes the resources found.
- **Ask**: Returns a boolean indicating whether a query pattern matches or not.

[16] http://139.91.183.30:9090/RDF/RQL/

Chapter 3: Web Personalization

3.1. Purpose for personalization and adaptation

In the past decade the web has massively grown and attracted many business domains, which made it attract many users. It has continued its growth to attract many users from different backgrounds, different needs and also different intuition and perception levels. Personalization tends to make web applications more user friendly by providing the correct or assumed to be correct needs of the user to facilitate the usage of the website or web application. The personalization process aims at providing users with small portions of the total data (which is massive) that exists on the web, provided that this portion is the part that meets the requirements and needs of the user. Personalized Web applications tailor the information presented and the structure to the user's preferences, knowledge or interests. Adaptive Web applications perform this adaptation dynamically learning from the user's navigation and interaction behaviour. The adaptation process varies according to the application type and the user needs and the level of flexibility the user should have. Examples of possible adaptation options are: the selection of pieces of information that are appropriate to the knowledge level of the user, some guidance performed through the removal of links that the system considers not relevant to the user, change in the data displayed or collected, and some recommendations given to the user. The system takes such decisions according to the knowledge it has of the user at given point in time [42]. Based on this knowledge most of the personalized applications perform a rule-based adaptation and include the management of a user profile. For example in an online book shop website, a first time user may be looking for books related to computer programming such as "Learning Computer Programming" the system could also recommend other books that are closely related to the same field, the system could suggest titles of other books such as "How Computer Programming Works", "The Art of Computer Programming" and "Elementary Computer Programming". If the system has a profile for that user and has knowledge of what programming languages the user knows, it then can recommend titles such as "Beginning Programming with Java For Dummies", and "Java Programming for the Absolute Beginner" (assuming the user knows java). This type of information about the user is stored in the system based on previous searches to determine the user preferences and thus be able predictions can be made about what he/she prefers. Of course this has to comply with the business logic of the application. This type of adaptation is called Content-based recommendations. Another approach is to provide the systems recommendations based on another user which had a similar case and what he preferred, this is called Collaborative recommendations. A Hybrid approach, which combines collaborative and content-based methods, could also be used [43]. This type of adaptation depends on rule based decision support, the business rules are formalized to logic formulas which the system uses to make the above scenario possible [44]. A question that pops in mind is: How accurate is this adaptation process? We can argue that the accuracy of adaptation depends greatly on the systems understanding of the user intensions and its ability to provide the appropriate set of rules that will fulfil the user needs. This is not always an easy task for the following reasons:

- **The system needs to be able to process the user needs based on the information it has**: The system does not understand what the user intentions are, meaning depends on the context it is said within. Therefore systems try to serve users based on the data that is available. In some cases the data available does not fulfil or partially fulfils the user needs. The user and the system have to have a shared semantics so that the relatedness of what the user wants is clearly understood by the system.

- **Users have higher expectations to how their goals are carried out**: A user looking for cheap plane tickets would expect the system to search the entire web for all available tickets. From the system point of view it is linked to some databases that contain plane tickets and starts its search from there. Once new sources of information are available the system has to be updated to maintain accurate results. This is due to the fact that knowledge is not provided globally for all systems, but rather such information would be maintained in databases, which are closed worlds.

3.1.1. Semantic Web Personalization and Adaptation

We could view the Semantic Web as a huge decentralized hypermedia system where the nodes represent data and metadata and the edges are the relations between the nodes. Can Semantic Web provide better adaptive systems? Semantic Web was developed with the intension of being machine readable and processable through combining the data with the semantics that describe what this data refers to through annotation and metadata. This provides the solutions for the adaptation problems mentioned above because it depends on understanding of semantics of the data and thus the user and the system share the same semantics. Some times a matching mechanism is required to unify user and system concepts. The annotation of data in Semantic Web and the use of ontologies for conveying meaning make applications closer to user needs and thus increases the accuracy of the adaptation and personalization process. The Semantic Web provides a platform where data is shared among applications and web applications. It breaks the tight coupling between the data and the application and thus frees the data from application, this comes from the fact that the Semantic Web is a web of data; the ability to express knowledge is the real power of the Semantic Web [45]. This allows for more cooperation and communication between applications, whether it is raw data (some general interpretation) or business data or both. Another aspect of personalization is the ability of the system to personalize the appearance of the application based on the user requirements. As an example a businessman may need to connect to a FOREX (Foreign Exchange) server from his PC and from his mobile, the presentation of the data in both cases is different. With the Semantic Web a recommender system adapted for this user will give him recommendation alerts for the pairs he is interested in; on the mobile or PC depending on his online state, each with a suitable presentation style; showing the risk, calculating the profit, and also when required providing proof of such claims. With Semantic Web this is possible because all data is annotated and applications share a common understanding of concepts, which reside in common ontologies and thus Semantic Web services and Semantic Web sites promise to achieve the high expectations of users. Imagine a scenario of a shopping agent, the user would personalize his agent specifying what he would like to buy, the upper limit of his budget, whether he is looking for a new or old item and whether he would want to buy online or locate the nearest store, etc. The shopping agent will take this information and start analyzing it, first it will reformalize the user query in terms of the vocabulary within the ontology or ontologies it is committed to, also it could do ontology matching for these terms with respect to other ontologies it is aware of. It will then search the knowledge bases for the matching instances and return the ones that satisfy the user query. This type of scenario is possible; first because the Semantic Web provides data in machine readable format and second the Semantic Web makes use of information by integrating it based on the semantics within the information.

The Semantic Web turns the web into one big knowledge base which gives promises of more accurate personalization and more accurate serving of user queries.

3.2. Semantic Web Personalization

The personalization process on the Semantic Web can be seen as a set of particular decisions about variable information sources. The Semantic Web contains a large amount of knowledge and will continue to grow. This knowledge is available for all systems on the Semantic Web to use, but this arises the problem of meeting the user needs. Even with this large amount of information a mechanism for information retrieval that fits the individual user requirements is required. This requires development of adaptive Semantic Websites that are able to provide users with only the portions of knowledge they require. Due to the fact that knowledge is self-descriptive on the Semantic Web, this adaptation process is thought to be richer and would most likely satisfy the user needs. In order to allow for making informed decisions, information intensive applications should on one hand collect and maintain knowledge about information resources and context of their usage, and user information on the other hand. Reasoning on the knowledge can then be applied to adapt access, presentation, and navigation in the information resources [48]. Personalization on the Semantic Web involves two steps: Information Collection and Retrieval, and Information Presentation [45].

1. **Information Collection and Retrieval**
   Semantic Websites are not document based like traditional websites; rather they retrieve their information based on variable distributed data sources, which are ontologies and knowledge bases. We can classify Semantic Websites to two types. The first type is websites that provide services such as a vacation planner or a medical agent or a shopping recommender, etc. This type would be linked to a main ontology, which defines its vocabulary and it will also search for and commit to other ontologies to retrieve similar and related terms. The second type is websites that provide services for a particular ontology, that ontology itself can be a combination of multiple ontologies. The need to find information from other existing ontologies on the Semantic Web will not be as massive as the first type and most probably they will be already imported in the main ontology the website is committed to; an example would be a museum website, a faculty

website and an E-Learning website. In both cases a Semantic Web query containing the user request, and taking into account his preferences regarding the retrieved data, has to be formulated by the system and executed by the query processor and reasoner.

2. **Information Presentation**
   Information representation is concerned with how the retrieved Semantic Web information will be displayed for the user. A Semantic Website (or any website) can be accessed through a multitude of devices and by different users. Each device has its own capabilities (display size, memory size, network speed etc.). Every user has specific preferences (desired layout, navigation patterns etc.) and browsing history with respect to a particular presentation. The website should take into account these constraints coming from both devices and users, and adapt the presentation accordingly [49]. Up to date there is no well agreement about how Semantic Web presentation should be implemented. However some proposals have been made on how Semantic Web data should be presented so that it provides the best visualization for the data and its taxonomy. Semantic Web browsers, also called Semantic Web User Interfaces (SWUI) [50] and XSL applied on the XML serialization of the Semantic Web data [51] are the most dominant options.

3.2.1. Our Proposed personalization model

When developing a personalized website one should take into account that the personalization process depends greatly on the understanding of the user needs, goals and objectives. The application logic is also an important feature each website has its own goals and thus the users that will use this website will have goals that interact with the site. For personalizing any website we must have knowledge of three fundamental things: The data (contents of the website), the user model (user intensions, preferences and goals), and the application logic (business logic). Based on who leads the adaptation process whether it is the system or the user, the application logic will contain rules and constraints that direct the system to how it should react. Knowing the context or domain of the website is very important, as it guides the system to what the users most probably mean by their queries and also gives the possibility of deducing various user interaction scenarios. Figure 14 gives our proposal for a Semantic Web personalization website, it contains three main modules: Query Generator, Semantic Web Framework, and Personalization Engine.

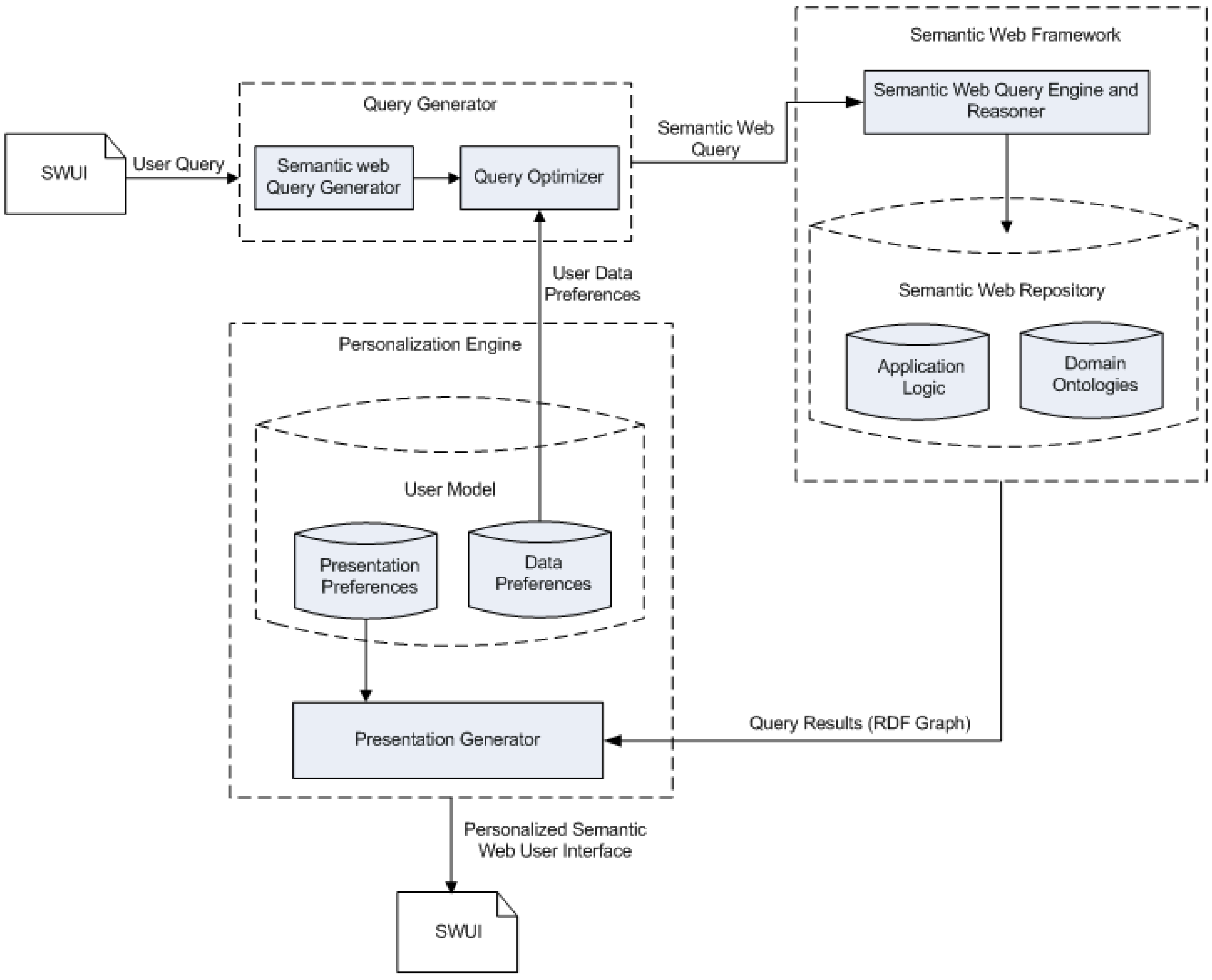


Figure 14: Our proposed Semantic Web Adaptation Model

Below is a detailed discussion of each component of our proposed Semantic Web adaptation model:

- **Query Generator:** it is composed of the Semantic Web query generator and the query optimizer. User queries are done in either implicit or explicit form. The user could enter his query in free text format and/or selects from existing options. On the other hand he could be navigating the system and looking for information on a certain topic. In either case this query is not in formal semantics format. The Semantic Web query generator transforms the query to the Semantic Web query based on its knowledge of the underlying domain ontology and application ontology which contains all the terms used within the application along with its semantics. The formalized query is a typical Semantic Web query that will retrieve the set of triples matching the search criteria. The query is semantically rich, and retrieves data based on the existent relationships within the knowledge base. Knowing the underlying knowledge model of the data, the query generator could define a set of rules for automatically creating the Semantic Web query based on the user query. The query optimizer then adds to the query generated, the user preferences regarding the data; this is extracted from the user model as shown in figure 14. The user preferences are added to the query based on a set of predefined rules by the adaptation system. It then combines this with the user query and optimizes the resulting query. The resulting query is a Semantic Web query formulated in one of the Semantic Web query languages (ex: RQL, SPARQL, SeRQL), which when executed returns the matching triples which can be represented by a Semantic Web graph (chapter 5 contains an example).

- **Semantic Web Framework:** it is composed of the Semantic Web query engine and reasoner, and the Semantic Web store which contains the data formatted in Semantic Web format. The repository contains at least two types of ontologies, domain ontologies which are general ontologies and the application logic ontology which contains all the business rules and constraints of the website. The reasoner would infer for new rules that come from combing both ontologies and then it will evaluate the query. It must be noted that the repository could also contain knowledge bases in case instances are kept separated. The query engine executes the query and returns the matching triples, which can be represented by a Semantic Web graph. If the query results are satisfying to the system, it will continue with the next step, which is displaying these results. If on the other hand the query results are not satisfying (do not meet a certain threshold for the number of outputs or there were no results found) the system will return back to the query generator. A new query will be generated based on the system recommendations and based on other user preferences if any exists.

- **Personalization Engine:** It is responsible for displaying the results within a Semantic Web user interface. It is composed of the user model and the presentation generator. The presentation preferences of the user are taken into account when displaying the query results. The presentation generator contains the necessary rules for displaying the data in a friendly user format. It also allows for user interaction with the data and thus submission of new queries for the system. This could be done using predefined templates that are filled with the query results. A user interface ontology should be defined to instruct the system to how the user interface should be generated. Another approach is using XSL transformation applied on the resulting triples after transforming them to RDF/XML serialization.

3.2.2. Including the user model in the adaptation process

In an adaptive system, the user model represents the system's assimilation of the interaction and contains information about the user and the current context that can increase the system's ability to exhibit pragmatically correct behaviour and, more generally, to engage in effective communication [52]. The user model could be a static user model which is created and changed by the user and the system uses it for the adaptation process. Or it could be a dynamic user model in which the initial user model could be created by the user and the system learns new preferences for that user by keeping track of the user behaviour on the site. The system could also predict new user behaviour based on previous users with similar preferences. The Model could also be entirely lead and updated by the system. In this case initially the user has no preferences but as he uses the site, the site will record his behaviour and extract from it his preferences. It is also important that the system verifies its deductions were correct by taking feedback from the user (customer satisfaction feedback). The user model should assist the system on how to provide information for the user through a set of rules that describe such preferences along with the user goals and intensions; this is also combined with the preferred communication devices used. Figure 14 showed how the user model could be used in the adaptation process. Extracting new user preferences is done via Web Mining techniques, which is out of the scope of this work.

3.3. Adaptation planning

Adaptation on the Semantic Web is different than the WWW that is why also the planning process is different. WWW is mainly about browsing documents, while Semantic Web is mainly about retrieval of knowledge. On the Semantic Web we are not only concerned with the appearance of the personalized system but also with the data behind it and how to get to this data. Semantic Web annotation provides the key issue here. The system must be built using fine-grained descriptive and logical items. System consistency comes from the consistency of its underlying ontologies and knowledge bases. The system should have a well-defined taxonomy for both the business rules and the domain rules; existing ontologies should be reused as much as possible. This will ensure that the integration of the user model data with the system easy and impose no logical contradictions. One of the challenges of personalization and adaptation is determining the features to personalize. The mission of the personalization process is to provide the user with the best way to do/ask for what he wants without feeling guided or restricted to only part of the information. Mainly there are two paradigms for personalization. In the first, it is either entirely left to the system to determine the user intensions and requirements and start acting based on this information, in this case it is called adaptation. In the second paradigm the user provides some information to the system, this information provides guidance for the system, this is called personalization. In both cases the system could get feedback on the user's satisfaction. When it comes to the presentation of the data the data should be organized in a way to maintain its semantics or its meaning. For example if the data represents a taxonomy, then this taxonomy should be clearly shown in the representation. In RDF there exist *label* and *language* terms which are mainly used for providing enhanced presentation of the data.

In Chapter 5 we give a case study to support our Semantic Web adaptation hypothesis.

Chapter 4: Semantic Web Tools

4.1. Introduction

Semantic Web development tools are essential for rapid and easy development of Semantic Websites. This chapter introduces some of the existing Semantic Web tools. There is a need for tools to support the development of ontologies and knowledge bases. Also storage tools to store and manipulate triples and finally tools to support query on the stored information. It must also be noted that there is a trend for standardization so these tools should offer standardized formats of RDF/RDFS and OWL. Quite a number of tools exist, such as [53][54][55]

- **Ontology Editors**: Protégé[17], DOE - The Differential Ontology Editor[18], IBM Ontology Management System[19] , IsaViz[20] and Altova's SemanticWorks[21], OntoStudio[22]
- **Large Triple stores**: Sesame[23], Kowari[24] and BigOWLIM [25]
- **Full development environments and servers:** RDF gateway[26], The Semantic Web Development Environment (SWeDE)[27]

Ontology editors should be able to validate the consistency of the edited ontology and report errors. Triple store on the other hand should be capable of storing and processing large amounts of triples and perform queries in the least time possible and allow for querying multiple ontologies. We will discuss two of these tools in more details namely: Protégé and Sesame.

---

[17] http://protege.stanford.edu/

[18] http://homepages.cwi.nl/~troncy/DOE/

[19] http://www.alphaworks.ibm.com/tech/snobase

[20] http://www.w3.org/2001/11/IsaViz/

[21] http://www.altova.com/products/semanticworks/semantic_web_rdf_owl_editor.html

[22] http://www.ontoprise.de/content/e1171/e1249/index_eng.html

[23] http://www.openrdf.com/

[24] http://www.kowari.org/

[25] http://www.ontotext.com/owlim/big/index.html

[26] http://www.intellidimension.com/

[27] http://owl-eclipse.projects.semwebcentral.org/

4.2. Protégé

The first step one needs to take when developing a Semantic Website is to identify and ontology editor to use. In this work we have used Protégé which is a free, open source ontology editor and knowledge-base framework. It allows visualization of ontologies and is extended by a large number of plug-ins [56]. The Protégé editor supports two main ways of modelling ontologies [57]:

- The Protégé-Frames editor enables users to build and populate ontologies that are frame-based, in accordance with the Open Knowledge Base Connectivity protocol [28] (OKBC). In this model, an ontology consists of a set of classes organized in a subsumption hierarchy to represent a domain's salient concepts, a set of slots associated to classes to describe their properties and relationships, and a set of instances of those classes.
- The Protégé-OWL editor enables users to build ontologies for the Semantic Web, in particular in the W3C's Web Ontology Language (OWL). Protégé in OWL mode enables the control of level of complexity it uses. It provides support for using OWL Lite , OWL DL and OWL full. It also allows for usage of RDF and RDF/RDFS

Protégé provides a highly scalable database back-end, allowing users to create ontologies with hundreds of thousands of classes. Protégé also supports saving an ontology in various formats namely RDF/XML N-triple N3. Protégé also has support for namespace usage and ontology reuse through allowing ontology import. When an ontology imports another ontology, all of the class, property and individual definitions that are in the imported ontology are available for use in the importing ontology, Protégé allows for building on these classes and properties and are shown in the ontology editor in read only mode. Protégé also has support for reasoning, users can download the reasoner they would like to use and configure Protégé to use it. This is possible because Protégé is a DIG aware application which enables it to use any DIG[29] (DL Implementation Group) reasoner such as: RACER, FaCT++ or any other DIG compliant reasoner [58].

[28] http://www.ai.sri.com/~okbc/

[29] http://dl.kr.org/dig/

In this work we have used Protégé to model an RDF/RDFS ontology via the Protégé Protégé-OWL editor. We used the import feature of Protégé to investigate the usage of sharing and reuse among ontologies. Ontoviz [59] visualization plug-in was used to visualize the created ontology.

4.3. Sesame

Sesame is an open source RDF database or RDF store (also called RDF repository) with support for RDF Schema inference and querying. It supports both HTTP access and SOAP access for applications to manipulate the RDF/RDFS data remotely or from the web. Internally it supports persistent storage of millions of triples, to do that it needs a scalable repository which is built on top of a DBMS. To be general and not dependent on a certain DBMS, all DBMS specific code is concentrated in a single architectural layer of Sesame called the Storage and Inference Layer. Sesame also supports not using a certain DBMS for the storage of the triples, in this case they are stored persistently in a file and Sesame deals with them as one big memory object, i.e. all operations are done in memory. This latter approach obviously decreases Sesame's performance a little but adds more flexibility to its usage [60]. Figure 15 shows the architecture of Sesame which consists of the following layers:

- **The Access APIs**: It provides direct access to Sesame's functional modules, either to a client program (for example, a desktop application that uses Sesame as a library), or to the next component of Sesame's architecture, which is the Sesame server. This is the component responsible for providing HTTP-based access to Sesame's APIs. Then, on the remote HTTP client side, we again find the access APIs, which are now used for communicating with Sesame not as a library, but as a server running on a remote location. [60]

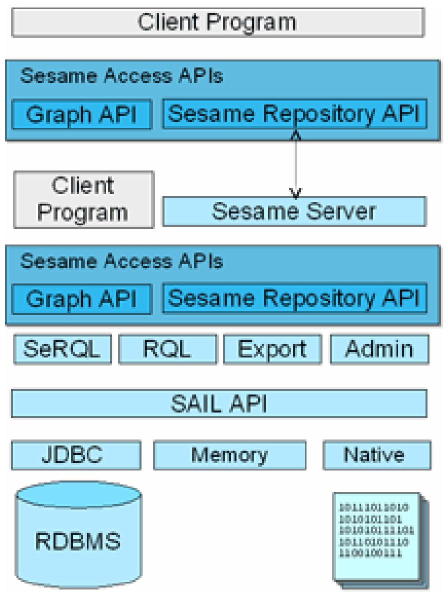


Figure 15: Sesame Architecture [8]

- **Admin:** This provides a program to facilitate performing administration operations. These operations include:
    - User management: allows setting user accounts and their permissions (Read/Write access).
    - Repository Management: allows configuration of existing repositories, adding new repositories and removing a repository.
    - Server configuration: allows loading a certain server configuration and allows the current configuration to be saved.

- **Export:** It provides export capability for the contents of a Sesame repository formatted in XML serialized RDF to other RDF representations namely: RDF/XML, N3, N-Triple and Turtle. This module allows extracting the data part only, the schema part only and both parts in the export operation. This eases the use of Sesame with other RDF tools.

- **RQL**: It is the RQL query engine, which allows for writing and executing RQL queries in Sesame. Sesame's RQL was implemented slightly different from the standard RQL language. The Sesame version of RQL features better compliance to W3C specifications, including support for optional domain- and range restrictions as well as multiple domain- and range restrictions. RQL queries are translated via the object model into a set of calls to the SAIL. This means that the main bulk of the actual evaluation of the RQL query is done in the RQL query engine itself [61].

- **SeRQL**: SeRQL or Sesame RDF Query Language is a query language developed especially for Sesame. It supports Construct and Select types of queries. A SeRQL query is typically built up from one to six clauses. For select queries these clauses are: *Select*, *From*, *Where*, *Limit*, *Offset* and *Using Namespace*, for construct queries only the first keyword is changed from Select to Construct. SeRQL query engine works with the same methodology as RQL by interacting with the SAIL [60].

- **The Storage and Inference Layer (SAIL)**: It is an application programming interface (API) that offers RDF specific methods to its clients and translates these methods to calls to its specific DBMS. This layer makes it possible to implement Sesame on top of a wide variety of repositories without changing any of Sesame's other components. Each Sesame repository has its own SAIL object to represent it. Sesame supports the following DBMS Currently, Sesame is able to use PostgreSQL, MySQL, Oracle and SQL Server [60][61].

In this work we have used Sesame as a repository for RDF and SPARQL as a query language for data retrieval.

Chapter 5: Semantic Website Personalization Use case

5.1. Introduction

In chapter 2 we have shown that Semantic Web is a web of knowledge. It makes it easy to express knowledge in a meaningful way. It also allows combing knowledge from more than one source to obtain richer knowledge. In chapter 3 we discussed the need of personalization and gave a proposal for a Semantic Web personalization system structure. To demonstrate how Semantic Web personalization works we did a small case study. We choose a cinema reservation system front end. By this case study we intend to show how Semantic Web enables us to write rich queries that make use of the relatedness of the underlying knowledge within the knowledge base. We will also show how we can use the knowledge about the user preferences (which resides in the user model) to create even richer queries that give the closest match for what the user requires.

5.2. Case Study

Let us assume we have a database that contains movies in the city with their exact time of start. The system would allow queries such as what time a certain film is displayed or what are the available shows for a certain film. But suppose that the users are interested to see a film which does not exist in this particular cinema they choose. Wouldn't it be useful if there was a query system rich enough and intelligent enough to suggest near by cinemas showing that particular film and with the display times that they prefer? Of course such knowledge would have to be accessible by the cinema reservation system application, and it is not restricted to it only. For example in another system we could be asking for the nearest post office or the nearest available pharmacies etc. This extra knowledge can be obtained from plugging in a Location ontology, which holds knowledge about spatial features of cities, districts, and streets. Having this knowledge in hand and combining it with the user preferences, for example knowing where he lives and what is the second best option for him, suggesting other cinema options that might be appealing for him, or suggesting other films with the same actors or actors with the same flavour, will enable more personalization options that are appealing to the users.

Our use case is implemented using Protégé as an ontology editor and Sesame as a repository and query engine. By this use case we aim at first constructing Semantic Web ontology that is rich enough to answer user queries. Second, construct and evaluate these user queries to obtain the set of triples that match the query constructed. For writing Semantic Web queries we have chosen SeRQL, introduced in chapter 4.

5.3. Proposed system Structure

5.3.1. Ontology Modelling

Analyzing domain knowledge is possible once a declarative specification of the terms is available. These specifications become clear in the context of the application and application domain required. Formal analysis of terms is extremely valuable when both attempting to reuse existing ontologies and extending them [62] [63]. When modelling an ontology one should keep in mind that an ontology maps knowledge of a certain domain independent on the application and independent on any special cases that may affect this knowledge representation. An ontology should not be tailored to any specific case. Ontology modelling implies modelling data, or objects in the UOD and modelling the axioms which represents the relations or role restrictions. There is no clear standard methodology for efficient creation of ontologies. In [62] they show that ontology modelling is an iterative process. It starts with defining the scope of the ontology and the domain it will conceptualize, the steps for ontology development according to [62] are:

1. **Determine scope:** This step answers the following questions: What is the domain that the ontology will cover? For what types of questions should the ontology provide answers?

2. **Consider reuse:** Search for an existing ontology that could contain fully or partially the terms that you require in your ontology or that model the same domain.

3. **Enumerate terms:** Write down an unstructured list all the relevant terms or lexicons that are expected to appear in the ontology. Typically, nouns form the basis for class names, and verbs form the basis for property names

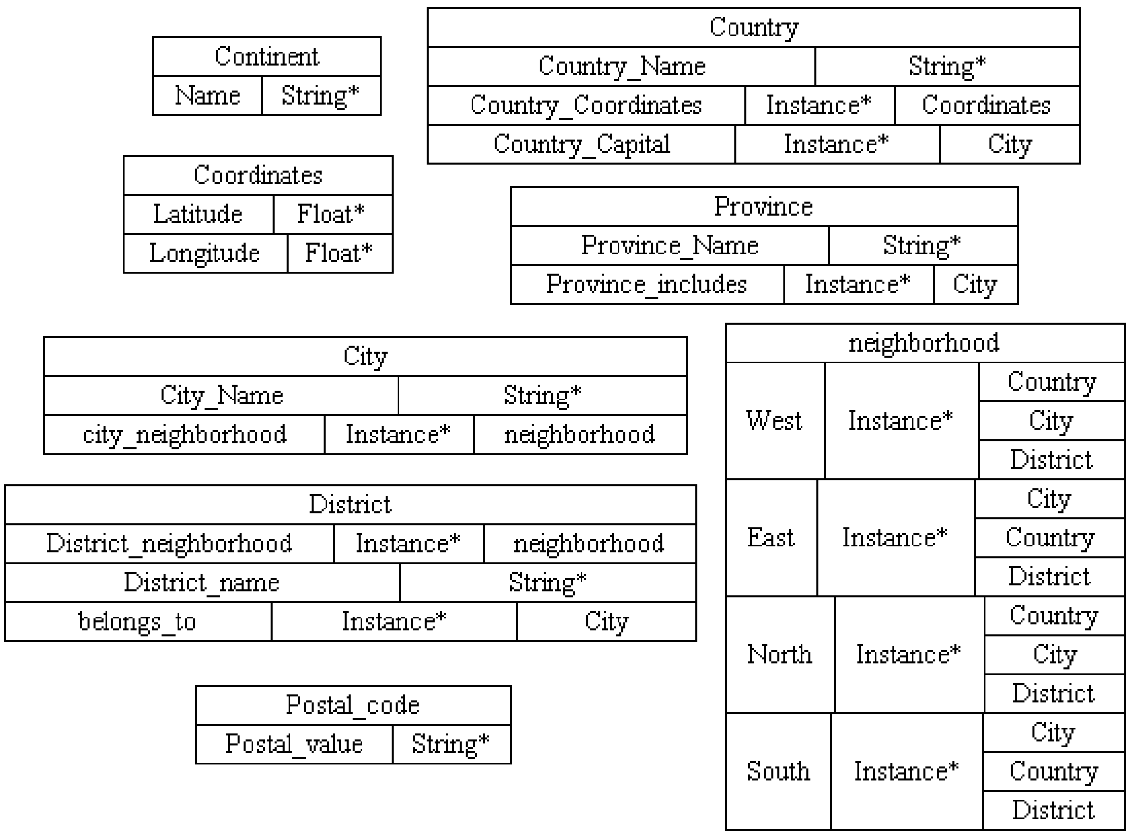
4. **Define taxonomy:** Define a hierarchy or hierarchies for the terms if any exists**.**
5. **Define properties:** Attach properties to classes, define the domain and range of each property.
6. **Define facets:** Add any constraints that exist to the properties or classes. At this point it will be possible to check the ontology for internal inconsistencies

5.3.2. Cinema Reservation System Modelling

As it is apparent from the problem specification, we need information about the films showed in cinemas: this is more like a time schedule for the films showed plus information about the film itself. Second, we have the information connecting films to each other based on the connections between the humans involved in the film making, namely the actors and the director. The ontology should express the relations between these parties and show their closeness. Third we have the spatial or location information about cities and districts which will be used to determine the nearness of alternative cinemas. From this discussion we could conclude that we need three ontologies:

- The Location Ontology
- The People Relationship Ontology
- The Film – Cinema Ontology

- **The Location Ontology:** This ontology maps the spatial relationships between cities and districts. Such an ontology does not exist so we have created a simple ontology which applies to all spatial relationships in any country, we have taken Belgium as a sample. We have stopped at the level of the district but the ontology can be extended to also include streets and any further divisions that exist. The location ontology also shows the relationships between the contents, countries and provinces and tries to give neighbourhood information between them. It also shows the coordinates of a certain country in terms of longitude and latitude values (this information could also be extended for the province and city). Figure 16 (taken from OntoViz visualization add-on in Protégé) shows the classes defined in the location ontology and their properties, for example the *Continent* class has one property called *Name* which is a string. The *neighbourhood* class has four properties: *East*, *South*, *North*, and *West* these properties are allowed to take values from *Country* ∪ *District* ∪ *City.* Figure 17 (taken from OntoViz visualization add-on in Protégé) shows the relationships between classes through expressing the domain and range of the properties (the arrow head refers to the range while the arrow start refers to the domain). For example the *country_capital* property has a domain *Country* and range *City*. This is a mapping for the statement, *a country has a capital, and the capital is one of its cities*. Another example is *south*, which has a domain *neighbourhood* and a *range* {*Country* ∪ *District* ∪ *City*}.



Continent	
Name	String*

Country		
Country_Name	String*	
Country_Coordinates	Instance*	Coordinates
Country_Capital	Instance*	City

Coordinates	
Latitude	Float*
Longitude	Float*

Province		
Province_Name	String*	
Province_includes	Instance*	City

City		
City_Name	String*	
city_neighborhood	Instance*	neighborhood

District		
District_neighborhood	Instance*	neighborhood
District_name	String*	
belongs_to	Instance*	City

Postal_code	
Postal_value	String*

neighborhood		
West	Instance*	Country
		City
		District
East	Instance*	City
		Country
		District
North	Instance*	Country
		City
		District
South	Instance*	City
		Country
		District

Figure 16: Location ontology classes and their properties

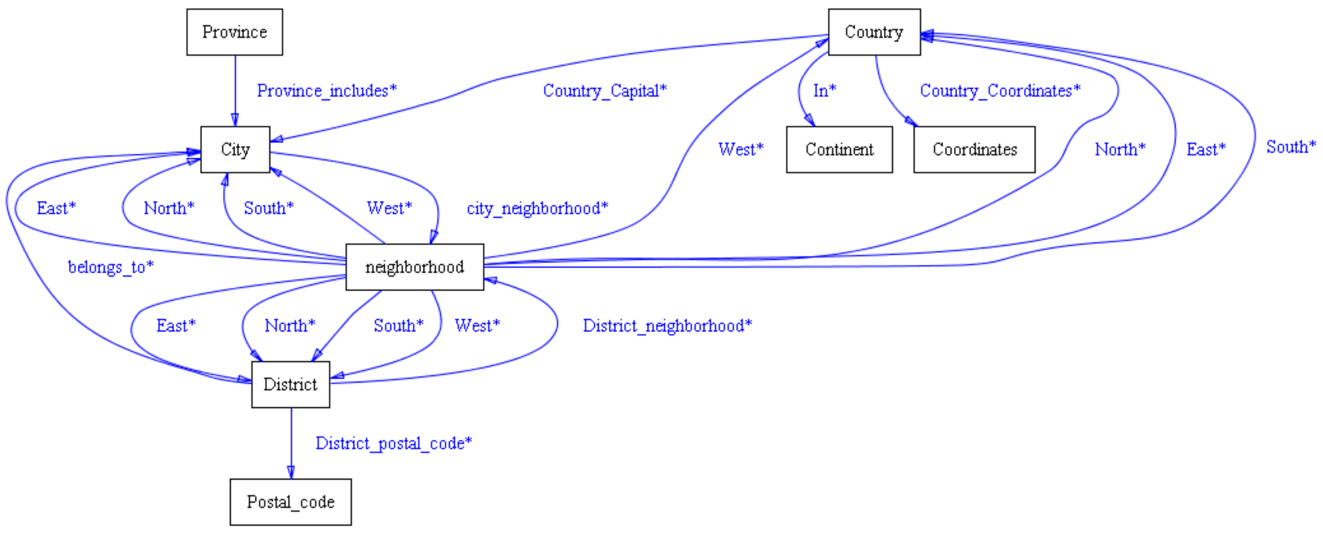


Figure 17: Location ontology relationships

- **The People Relationship Ontology:** We need to map the relationships between actors, actresses and directors. These relationships are relationships among people or persons. This is why we decided to use the FOAF[30] (Friend Of A Friend) ontology, which is a very general ontology to describe persons. ***"The Friend of a Friend (FOAF) project is creating a Web of machine-readable pages describing people, the links between them and the things they create and do"*** [64].

  FOAF is described in RDF syntax, and for generalization and reuse sake the FOAF classes have been related to their equivalents in other ontologies. This allows FOAF data to be immediately processable by applications built to understand these ontologies. For example FOAF refers to and uses concepts from WordNet[31] ontology, Contact[32] ontology, and Geographic[33] ontology [65][66]. Figure 18 shows the FOAF class hierarchy. Where p1 is the namespace for the WordNet ontology, p2 is the namespace for the Contact ontology and p3 is the namespace for the Geographic ontology.

[30] http://xmlns.com/foaf/0.1/

[31] http://xmlns.com/wordnet/1.6/

[32] http://www.w3.org/2000/10/swap/pim/contact

[33] http://www.w3.org/2003/01/geo/

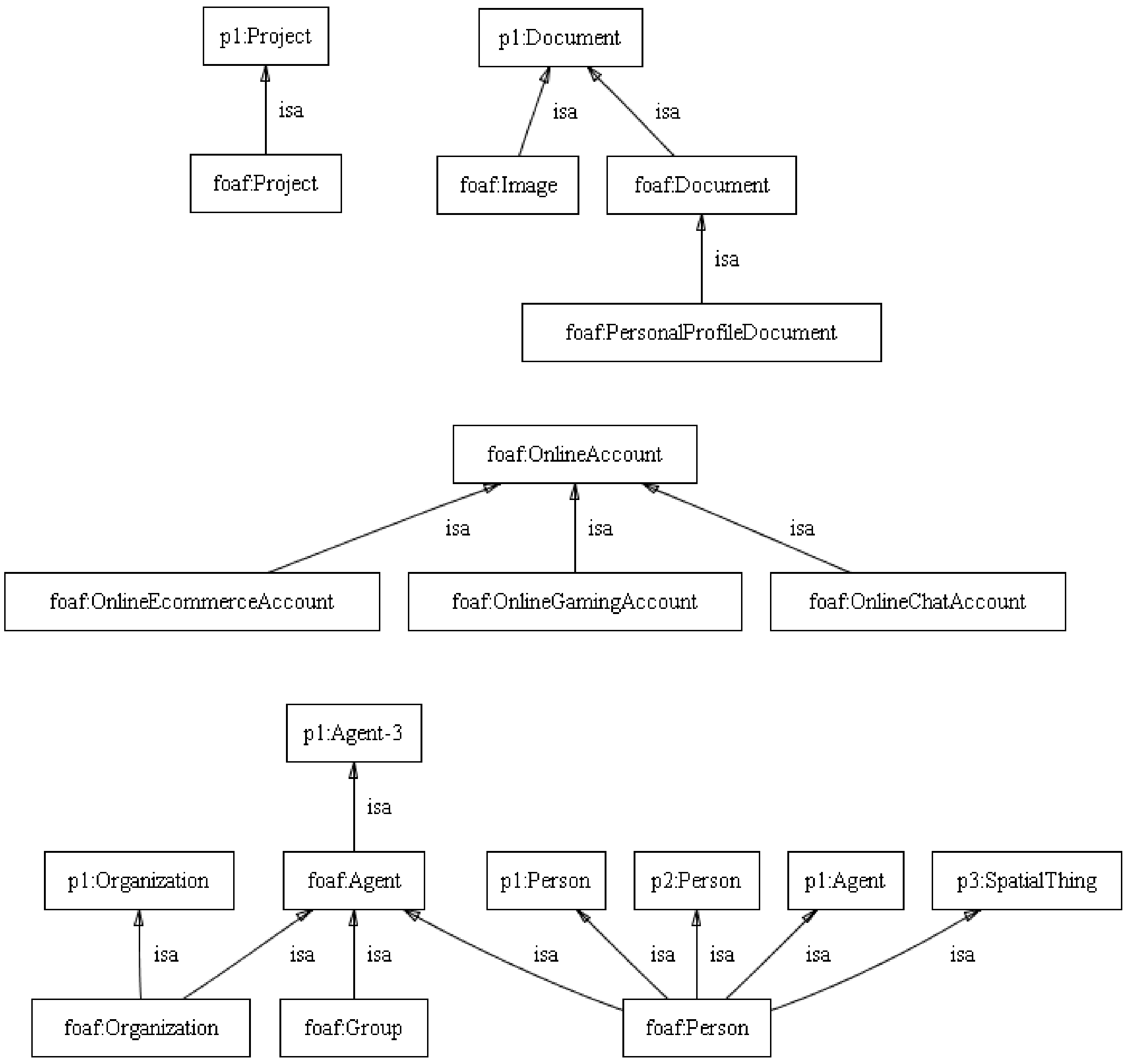


Figure 18: FOAF class hierarchy

The Basic class in FOAF is the *person* class which maps persons, some of its properties are: *name*, *firstname*, *nick*, *homepage*, *image*, *knows*, *pastproject*, *mbox*. *Knows* refers to another person, whom this person knows. *Pastproject* refers to the past projects of this person. *Mbox* refers to the mailbox of this person. In our case study, the three classes: *actor*, *actress* and *director* inherit the FOAF person class. In this case all the properties of the person class are also available for these three classes. In that case *knows* property indicates other actors/actress that this person worked with. The *pastproject* property indicates his past films. Figure 19 shows this inheritance hierarchy. Figure 20 shows the Actor, Actress, and Director classes within the FOAF class hierarchy.

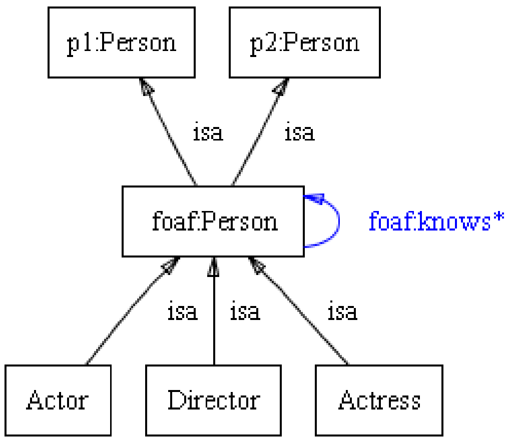
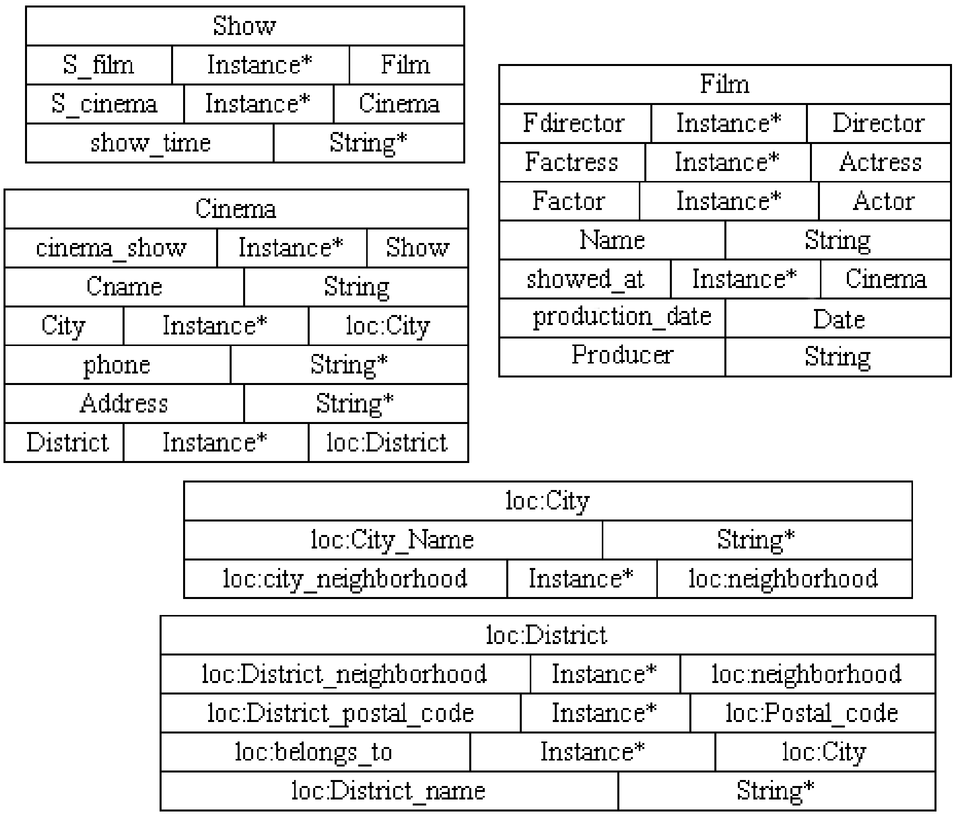


Figure 19: Actor, Actress, and Director within the FOAF class hierarchy

- **The Film-Cinema ontology:** This ontology maps the Film information, the cinema information and the relationship between the film and cinema which is defined within the show information. It has namely 3 main classes: Film, Cinema, and Show. Film represents the film, it has as properties *Name* which refers to the film name, *Factor* which refers to the film actor(s), *Factress* which refers to the film actress(s), *Producer* which refers to the film producer, *Director* which refers to the film director(s), Production_date and *showed_at*. *Production_date* is a date referring to the *xsd:date* and *showed_at* refers to the cinema at which the film is showed. Cinema represents the available cinemas, it has properties *Cname* which refers to the name of the cinema, *City* which refers to the city that the cinema is located in – this term is taken from the location ontology, *District* which refers to the district where the cinema resides- also taken from the location ontology, *phone* which refers to the phone number of the cinema, and *Address* which refers to the address of the cinema (street address).

  Figure 20 shows the Cinema ontology classes and their properties. The District and City classes are part of the Location ontology and are given a namespace *loc*. Figure 21 shows the domain and range of the properties of the Film-Cinema ontology. Figure 22 shows the class hierarchy of the Film-Cinema ontology after importing both the FOAF and the Location ontologies.



Show		
S_film	Instance*	Film
S_cinema	Instance*	Cinema
show_time	String*	

Film		
Fdirector	Instance*	Director
Factress	Instance*	Actress
Factor	Instance*	Actor
Name	String	
showed_at	Instance*	Cinema
production_date	Date	
Producer	String	

Cinema		
cinema_show	Instance*	Show
Cname	String	
City	Instance*	loc:City
phone	String*	
Address	String*	
District	Instance*	loc:District

loc:City		
loc:City_Name	String*	
loc:city_neighborhood	Instance*	loc:neighborhood

loc:District		
loc:District_neighborhood	Instance*	loc:neighborhood
loc:District_postal_code	Instance*	loc:Postal_code
loc:belongs_to	Instance*	loc:City
loc:District_name	String*	

Figure 20: Film-Cinema ontology classes and their properties

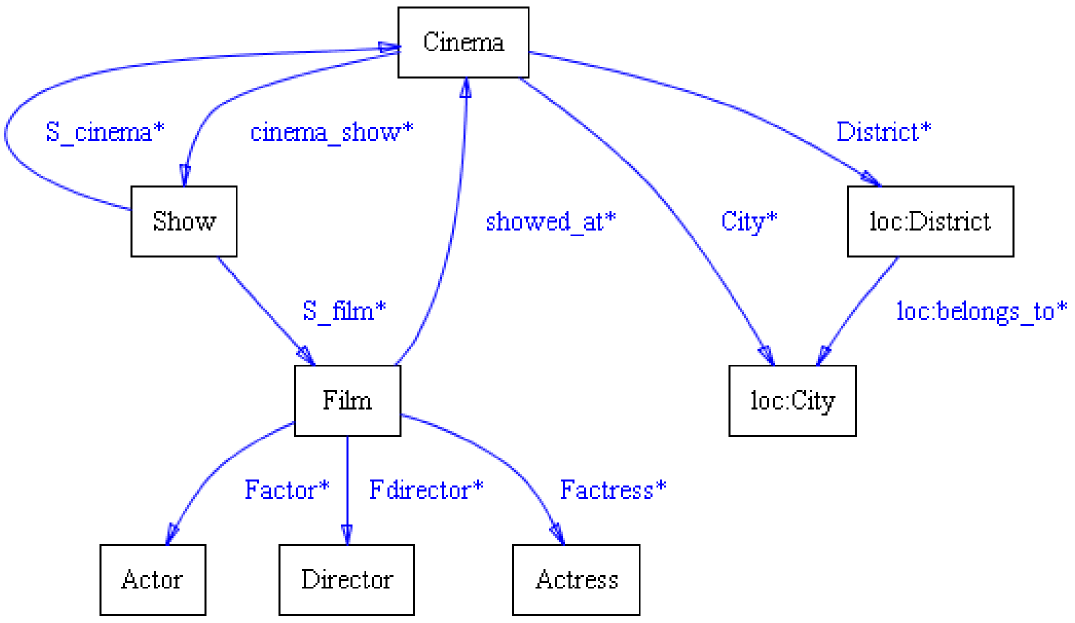


Figure 21: Film-Cinema ontology relationships

It must be noted that when importing an ontology all its classes, properties and axioms are available and also all its instances. So now Film-Cinema becomes our big repository where all the knowledge is stored. Adding instances to it makes it become our main knowledge base. In the next section we will show how to query this knowledge base to extract information form it.

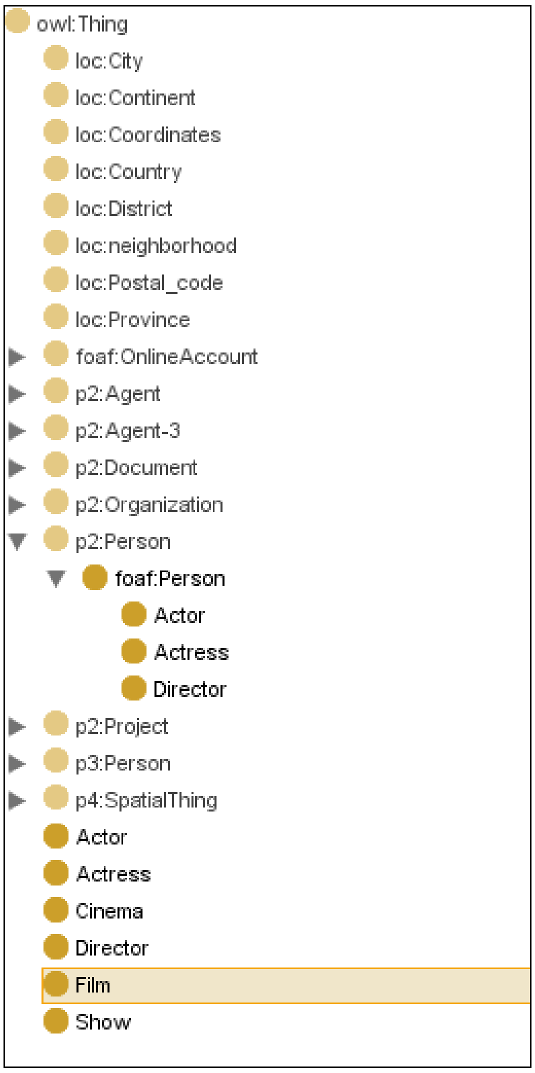


Figure 22: Class hierarchy within the Film- Cinema ontology, Location ontology and FOAF ontology included

5.4. Creating Richer queries

Now that the data is in the knowledge base, we should be able to execute SERQL queries to retrieve this data. These queries should take into consideration the knowledge base model which the queries are executed against. Thus querying RDF is done based on the existing triples and their relations.

We will demonstrate this ides through the following set of queries:

Query 1: If a user is searching for the available cinemas, that display the film *Take the Lead* this query is shown in listing 6:

```
select  film, cinema,time
from {film}  ns1:Name {fname},{show} ns1:S_cinema {cinema},{show}
ns1:S_film {film},{show} ns1:show_time {time}

where fname like "Take the Lead"

using namespace
 ns1=<http://localhost/film#>
```

Listing 6: Query 1

film	cinema	time
http://localhost/film#F1	http://localhost/film#C3	"2006-08-11T15:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C8	"2006-08-11T21:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C3	"2006-08-12T21:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C7	"2006-08-12T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C8	"2006-08-11T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C7	"2006-08-11T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime

http://localhost/film#F1	http://localhost/film#C1	"2006-08-12T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C1	"2006-08-13T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C3	"2006-08-12T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C3	"2006-08-12T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C5	"2006-08-11T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C1	"2006-08-11T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C7	"2006-08-11T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C1	"2006-08-12T21:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C1	"2006-08-11T21:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C1	"2006-08-11T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C8	"2006-08-11T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C3	"2006-08-11T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C1	"2006-08-12T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C5	"2006-08-12T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C1	"2006-08-12T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C7	"2006-08-12T21:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C7	"2006-08-

		12T15:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C5	"2006-08-11T21:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C8	"2006-08-12T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
http://localhost/film#F1	http://localhost/film#C8	"2006-08-12T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime

26 results found in 230 ms.

Table 2 : Query 1 Results

In building this query we have to follow the relations between the triples, and we make a chain of triples based on what the required output is. Query 1 can be extended to ask for the cinemas in Leuven that display that film. This knowledge is possible from the *City* property in the cinema class. In this case the relationship of City must be taken into consideration in the query and thus we refer to the Location ontology City to retrieve the *City_Name*. Query 2 in listing 7 shows this extension.

```
select  film, cinema,time,ccname

from {film}  ns1:Name {fname},{show} ns1:S_cinema {cinema},{show}
ns1:S_film {film},{show} ns1:show_time {time}, {cinema} ns1:City
{cname},{cname} ns2:City_Name {ccname}

where fname like "Take the Lead" and ccname like "Leuven" IGNORE CASE

using namespace
 ns1=<http://localhost/film#>,
ns2=<http://localhost/location#>
```

Listing 7: Query 2

film	cinema	time	ccname
http://localhost/film#F1	http://localhost/film#C7	"2006-08-12T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Leuven"@en
http://localhost/film#F1	http://localhost/film#C7	"2006-08-11T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Leuven"@en
http://localhost/film#F1	http://localhost/film#C5	"2006-08-11T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Leuven"@en
http://localhost/film#F1	http://localhost/film#C7	"2006-08-11T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Leuven"@en
http://localhost/film#F1	http://localhost/film#C5	"2006-08-12T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Leuven"@en
http://localhost/film#F1	http://localhost/film#C7	"2006-08-12T21:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Leuven"@en
http://localhost/film#F1	http://localhost/film#C7	"2006-08-12T15:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Leuven"@en
http://localhost/film#F1	http://localhost/film#C5	"2006-08-11T21:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Leuven"@en

8 results found in 601 ms.

Table 3: Query 2 results

Suppose the user wishes to know where exactly in Leuven the cinemas that display that film are. In this case we have to plug-in the information from the *District* and the *address* properties of the *cinema* class. To retrieve the district name we need to refer to the Location ontology and retrieve the *District_name* property of the *district* class. This is shown in query 3 in listing 8.

```
select  film, cinname,time,dname, address
from {film}  ns1:Name {fname},{show} ns1:S_cinema {cinema},{show} ns1:S_film
{film},{show} ns1:show_time {time}, {cinema} ns1:City {cname},{cname}
ns2:City_Name {ccname}, {cinema}ns1:Cname {cinname},  {cinema}ns1:District
{district},{district} ns2:District_name {dname}, {cinema} ns1:Address {address}

where fname like "Take the Lead" and ccname like "Leuven" IGNORE CASE

using namespace
 ns1=<http://localhost/film#>,
ns2=<http://localhost/location#>
```

Listing 8: Query 3

film	cinname	time	dname	address
http://localhost/film#F1	"Cinema Zed"^^http://www.w3.org/2001/XMLSchema#string	"2006-08-12T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Heverelee"@en	"Naamsestraat, 96"^^http://www.w3.org/2001/XMLSchema#string
http://localhost/film#F1	"Cinema Zed"^^http://www.w3.org/2001/XMLSchema#string	"2006-08-11T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Heverelee"@en	"Naamsestraat, 96"^^http://www.w3.org/2001/XMLSchema#string
http://localhost/film#F1	"Studio Leuven"^^http://www.w3.org/2001/XMLSchema#string	"2006-08-11T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Kessel-lo"@en	"Burgemeesterstraat, 30"^^http://www.w3.org/2001/XMLSchema#string
http://localhost/film#F1	"Cinema Zed"^^http://www.w3.org/2001/XMLSchema#string	"2006-08-11T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Heverelee"@en	"Naamsestraat, 96"^^http://www.w3.org/2001/XMLSchema#string
http://localhost/film#F1	"Studio Leuven"^^http://www.w3.org/2001/XMLSchema#strin	"2006-08-12T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTi	"Kessel-lo"@en	"Burgemeesterstraat, 30"^^http://www.w3.org/2001/XMLSchema#strin

	g	me		g
http://localhost/film#F1	"Cinema Zed"^^http://www.w3.org/2001/XMLSchema#string	"2006-08-12T21:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Heverelee"@en	"Naamsestraat, 96"^^http://www.w3.org/2001/XMLSchema#string
http://localhost/film#F1	"Cinema Zed"^^http://www.w3.org/2001/XMLSchema#string	"2006-08-12T15:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Heverelee"@en	"Naamsestraat, 96"^^http://www.w3.org/2001/XMLSchema#string
http://localhost/film#F1	"Studio Leuven"^^http://www.w3.org/2001/XMLSchema#string	"2006-08-11T21:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime	"Kessel-lo"@en	"Burgemeesterstraat, 30"^^http://www.w3.org/2001/XMLSchema#string

8 results found in 91 ms.

Table 4: Query 3 results

We can argue from the three sample queries we gave that the Semantic Web makes it easier to construct rich queries and extend these queries based on the underlying knowledge model. It is also easy to automatically generate these extensions based on the understanding of the model structure. Also at any given time in the system once new knowledge is plugged in we can start querying the knowledge base for information which could come from multiple sources. This was clear in query 2 and query 3 where we retrieved knowledge from the Location ontology along with the film ontology. Of course this could be extended to any number of related ontologies.

5.5. Using the user model for query result personalization

In the above section we investigated how a Semantic Web query can be formed and extended to give adapted results based on the system requirements or known in advance requirements. But in user-adaptive systems is it usually the case where at run time the system needs to generate queries that take into account the user preferences. These preferences are normally stored in the user model. As an example a user searching for the film *Take the Lead* in Leuven and there was no matches found. What the system will do is based on the user model recommend other cinemas in the neighbourhood of Leuven or recommend other films that are displayed in Leuven. Listing 9 shows the query in the case where the user prefers to search for a cinema in the neighbourhood districts that is displaying his selected film. Where *cinname* is the cinema name, *address* is the cinema address and *ddname* is the district name.

```
select distinct   cinname,address,ddname

from {film}  ns1:Name {fname},{show} ns1:S_cinema {cinema},{show}
ns1:S_film {film},{show} ns1:show_time {time},  {cinema} ns1:City
{cname},{cname} ns2:City_Name {ccname}, {cinema}ns1:Cname {cinname},
{district} ns2:District_name {dname}, {cinema} ns1:Address
{address},{district} ns2:District_neighborhood {nbh}, [{nbh} ns2:South
{d1},{cinema}ns1:District {d1},{d1} ns2:District_name {ddname} ],
[{nbh} ns2:East {d2},{cinema}ns1:District {d2},{d2} ns2:District_name
{ddname}], [{nbh} ns2:West {d3},{cinema}ns1:District {d3},{d3}
ns2:District_name {ddname}], [{nbh} ns2:North
{d4},{cinema}ns1:District {d4}, {d4} ns2:District_name {ddname}]

where fname like "Take the Lead"  and ccname like "Leuven"  and dname
like "leuven" IGNORE CASE

using namespace
 ns1=<http://localhost/film#> ,
ns2=<http://localhost/location#>
```

Listing 9: query retrieving cinemas in the neighboorhood districts

cinname	address	ddname
"Cinema Zed"^^http://www.w3.org/2001/XMLSchema#string	"Naamsestraat, 96"^^http://www.w3.org/2001/XMLSchema#string	"Heverelee"@en
"Studio Leuven"^^http://www.w3.org/2001/XMLSchema#string	"Burgemeesterstraat, 30"^^http://www.w3.org/2001/XMLSchema#string	"Kessel-lo"@en

2 results found in 551 ms.

Table 5: Query results

Listing 10 shows the other option, this is that the user prefers a cinema in Leuven, and would like to see what films are showing there.

```
select    fname,cinname,address,time
from {film}  ns1:Name {fname},{show} ns1:S_cinema {cinema},{show}
ns1:S_film {film},{show} ns1:show_time {time},  {cinema} ns1:City
{cname},{cname} ns2:City_Name {ccname}, {cinema}ns1:Cname {cinname},
{cinema}ns1:District {district},{district} ns2:District_name {dname},
{cinema} ns1:Address {address}

where ccname like "Leuven"  and dname like "leuven" IGNORE CASE

using namespace
 ns1=<http://localhost/film#> ,
ns2=<http://localhost/location#>
```

Listing 10: Query retrieving other films available in cinemas in Leuven

fname	cinname	address	time
"Zoom"^^http://www.w3.org/2001/XMLSchema#string	"Kinepolis Leuven"^^http://www.w3.org/2001/XMLSchema#string	"Bondgenotenlaan, 145-149"^^http://www.w3.org/2001/XMLSchema#string	"2006-08-12T21:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
"Zoom"^^http://www.w3.org/2001/XMLSchema#string	"Kinepolis Leuven"^^http://www.w3.org/2001/XMLSchema#string	"Bondgenotenlaan, 145-149"^^http://www.w3.org/2001/XMLSchema#string	"2006-08-12T10:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
"World Trade Center"^^http://www.w3.org/2001/XMLSchema#string	"Kinepolis Leuven"^^http://www.w3.org/2001/XMLSchema#string	"Bondgenotenlaan, 145-149"^^http://www.w3.org/2001/XMLSchema#string	"2006-08-12T21:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime
"World Trade	"Kinepolis	"Bondgenotenlaan,	"2006-08-

Center"^^http://www.w3.org/2001/XMLSchema#string	Leuven"^^http://www.w3.org/2001/XMLSchema#string	145-149"^^http://www.w3.org/2001/XMLSchema#string	12T18:00:00"^^http://www.w3.org/2001/XMLSchema#dateTime

4 results found in 10 ms.

Table 6: Query results

5.6. Results and discussion

From the design and implementation of this case study we can conclude that the Semantic Web provides a richer platform for data representation. Semantic Web metadata languages enable the well description of knowledge. Through the modelling phase it is important to take into consideration the reuse of existing ontologies and carefully plan the knowledge model. We have demonstrated how ontologies can be plugged in together to create a larger world via the import facility in OWL. Further more classes can inherit other classes in these imported ontologies to provide extended features.

We have shown that the Semantic Web enables rich queries to be made. These queries are based on the knowledge model which maps the semantics of the data. Once the model is known, creating a query is like traversing the graph edges to retrieve these edges that are beneficial for your query. The system can start by creating a small query and then extending this query based on the user requirements or user actions. It is clear from our sample queries that knowing the relations between the triples is all that is required and the linkage between the data is done either via a common subject or common predicate or common object within the triple. Further more we have shown that combining the knowledge about the user preferences and taking it into account to create a more dynamic level of adaptability is easily done in Semantic Web. User preferences are mapped into a selection of triples to take into consideration within the query. This enables the creation of these rich queries according to predefined rules in the system which generates on the fly adaptation suitable to different user models. We think that representing the user preferences should also be placed in a user model ontology which will make it easier for the system to plug-in the user preferences in the query formulation process.

Chapter 6: Results and Recommendations

6.1. Summery

In this thesis we have studied the Semantic Web and the possibilities of Semantic Web adaptation. We have proposed a system architecture for such adaptation process (chapter 3). The model depends on the query formalization and user preferences obtained from the user model. The Semantic Web formalized query is evaluated against the knowledge base repository. At this stage the query engine is responsible for inferring new knowledge relevant for the query results. We also did a case study to verify that the Semantic Web enables both rich representation of knowledge and rich query composition for knowledge retrieval (chapter 5). Generating the Semantic Web query knowing the underlying knowledge model is easily done from the relations between the triples. It is a process of traversing the Semantic Web graph, with only significant triples for the query being taken into account. Adaptation to user requirements is easily done by taking into consideration the user preferences while determining which triples are relevant. Different user requirements will lead to different requested triples in the query (chapter 5). We have also shown that the Semantic Web is a web of data which is very different from the WWW (chapter 2). In the Semantic Web knowledge representation with metadata languages (RDF/RDFS - OWL) makes it possible to represent relations between the data in a machine readable format. The complexity of the data representation or the complexity of the knowledge model is determined by the metadata language used. Semantic Web languages provide the option to model knowledge in a simple form – using RDF, and also in a complex one – using OWL full. Providing knowledge in this manner makes it possible to combine knowledge from more than one source and reuse it (discussed in chapter 2 and demonstrated in chapter 5). This feature turns the Semantic Web to one big knowledge base where agents or applications can efficiently use the knowledge made available on the Semantic Web. Any ontology or knowledge base that is reachable via the internet is available for sharing and reuse by other knowledge bases and ontologies. The knowledge model or the ontology should be specific to a certain domain to keep it as general as possible, for ease of reusability among different applications (chapter 2). We have also discussed searching on the Semantic Web (chapter 2), for end users and also researchers searching is a very important process. We have shown that search on the Semantic Web is very different from the case of document traversing of the WWW. Semantic Web search should make great emphasis on the semantics of the search keywords and the desired user search output. We gave examples of three existing search engines, each having a different working methodology. One of which is Swoogle, which is currently the most successful Semantic Web search engine on the web. Currently there is quite a number of Semantic Web tools that enable editing and generation of ontologies. Also tools for querying and reasoning, and storing the triples. We also gave an overview of some of the existing tools and focused on Protégé and Sesame (chapter 4), which we used for implementing our case study (chapter 5).

6.2. Conclusion and further recommendations

The Semantic Web promises to bring meaning to the knowledge. This opens the way for new intelligent applications and services, which understand the semantics of the data and thus provide the user with a better service. Semantic Web serves as a global knowledge base. This globalization of knowledge along with the fact that the knowledge is self describable and thus understandable, is what we think is the source of power for the Semantic Web and will lead to its continuity and success. This opens the way for more user oriented adaptive systems that will request and retrieve knowledge based on the user progress within the system (in case of E-Learning applications) or based on the requests and preferences made by the user (in case of E-business solutions and recommender systems). Semantic Web makes the process of retrieval of user tailored knowledge an easy and more accurate task. Thus it fully supports the creations of system driven adaptation systems. We have shown that creating Semantic Web queries that retrieve knowledge is a very scalable process and can easily be automated once the knowledge model is known. User-driven adaptation is also improved through the understanding of what the user requires and linking that to the knowledge model. We gave a proposal for the structure of a Semantic Web adaptive system, which takes into consideration both the user model and the system knowledge model. We can conclude from our results that the following points:

1. Semantic Web success in representing knowledge depends greatly on knowledge modelling. There should be careful modelling of ontologies and knowledge bases to enable rich knowledge representation.

2. Semantic Web adaptation is based on the systems understanding of the underlying knowledge and thus its ability to retrieve only the knowledge that fits the user needs.
3. Semantic Web enable the creation of semantically rich and scalable queries for data retrieval. It also enables inferring new semantics of the data when combining multiple Semantic Web knowledge sources.
4. When it comes to combining knowledge from many sources on the web, the layers of proof and trust become very important in verifying the consistency of the data.

We also recommend the following points for future research and investigation:

1. We did not study the impact of adaptation on presentation (in the user interface or browser) of the Semantic Web data. This point is quite important because it reflects how the end user will deal with the system. We proposed it as part of the adaptation model and think it needs further investigation.
2. We argue that the Semantic Web query can easily be built automatically by defining (configuring) the rules to include or exclude triple sets from the query. We think this will enable the generation of adaptive systems that are easy to implement and maintain, a further research on this issue will be interesting.
3. User-oriented adaptation systems learn from the user requirements and behaviour with the system to enhance the adaptation process to better fit the needs of the user. This feedback we think is important and will open the way to dynamic and yet user appealing user-adaptation. This point should be investigated as future work to enhance user-adaptive systems performance.
4. There is a need for more precise methodology for ontology creation. Also research communities should give great importance to creating shared public ontology repositories in different domains.

References

1. Tim Berners-Lee, Jim Hendler, and Ora Lassila. The Semantic Web. Scientific American, 2001, http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C7084A9809EC588EF21&pageNumber=2&catID=2

2. Semantic web Activity, W3C, http://www.w3.org/2001/sw/

3. Graham Klyne and Jeremy J. Carroll, Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C http://www.w3.org/TR/rdf-concepts

4. Tim Berners-Lee and Eric Miller, 'The Semantic Web lifts off'. ERCIM News No. 51, October 2002.

5. Tim Berners-Lee, Semantic Web - XML2000, talk in 2000, http://www.w3.org/2000/Talks/1206-xml2k-tbl/Overview-2.html

6. Grigoris Antoniou and Frank van Harmelen, A Semantic Web Primer, MIT Press , April 2004

7. Frank Manola and Eric Miller, RDF Primer, W3C, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/ , February 2004

8. Dave Beckett, RDF/XML Syntax Specification, W3C, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/, February 2004

9. Tim Bray, Dave Hollander and Andrew Layman, Namespaces in XML, W3C, http://www.w3.org/TR/1999/REC-xml-names-19990114, 1999

10. David C. Fallside and Priscilla Walmsley, XML Schema Part 0: Primer Second Edition, W3C, http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/ , 2004

11. Dan Brickley and R.V. Guha, RDF Vocabulary Description Language 1.0: RDF Schema, W3C, http://www.w3.org/TR/rdf-schema/, 2004

12. Tim Berners-Lee, Semantic Web Road map, W3C , http://www.w3.org/DesignIssues/Semantic.html, September 1998

13. Marja-Riitta Koivunen and Eric Miller, W3C Semantic Web Activity, Semantic Web Kick-off Seminar in Finland Nov 2, 2001

14. Primer: Getting into RDF & Semantic Web using N3, http://www.w3.org/2000/10/swap/Primer

15. Dave Beckett, N- Triple, http://www.dajobe.org/2001/06/ntriples/ , 2001

16. M. Genesereth and N. Nilsson, "Logical Foundations of Artificial Intelligence", Morgan-Kaufmann Publications Inc., 1988

17. Aditya Kalyanpur, Jennifer Golbeck, Jay Banerjee, and James Hendler, Owl: Capturing semantic information using a standardized web ontology language, Multilingual Computing & Technology Magazine, Vol. 15, issue 7, Nov 2004.

18. Peter F. Patel-Schneider and Ian Horrocks, OWL Web Ontology Language Semantics and Abstract Syntax, Section 2. Abstract Syntax, W3C, 2004, http://www.w3.org/TR/2004/REC-owl-semantics-20040210/syntax.html

19. Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider and Lynn Andrea Stein, OWL Web Ontology Language Reference, W3C, 2004, http://www.w3.org/TR/owl-ref/

20. Deborah L. McGuinness and Frank van Harmelen, OWL Web Ontology Language Overview, W3C, 2004, http://www.w3.org/TR/owl-features/

21. Jeff Heflin, OWL Web Ontology Language Use Cases and Requirements, W3C, 2004 http://www.w3.org/TR/webont-req/

22. Gruber, T.R, A Translation Approach to Portable Ontologies. J. on Knowledge Acquisition, Vol. 5(2), 199 -220 (1993).

23. Gruber, T. 1995. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal of Human and Computer Studies, 43(5/6): 907-928.

24. Dieter Fensel, Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, Springer 2001

25. CMS Wiki , http://www.cmswiki.com/tiki-index.php?page=Ontology

26. Tim Berners-Lee, Semantic Web: Building on What Exists, , MIT Information Technology Conference keynote, February 2006, http://www.w3.org/2006/Talks/0314-ox-tbl

27. Intellidimension, Semantic Web Search - A Next Generation Internet Search Engine, http://www.semanticwebsearch.com/news/item.rsp?n=2

28. Danny Sullivan, How Search Engines Work, 2002, Search Engine Watch, http://searchenginewatch.com/webmasters/article.php/2168031

29. Sergey Brin and Lawrence Page, The Anatomy of a Large-Scale Hypertextual Web Search Engine, WWW7 / Computer Networks, 1998

30. Li Ding, Tim Finin, Anupam Joshi, Yun Peng, R. Scott Cost, Joel Sachs, Rong Pan, Pavan Reddivari and Vishal Doshi , Swoogle: A Semantic Web Search and Metadata Engine, Proceedings of the Thirteenth ACM Conference on Information and Knowledge Management, 2004

31. Li Ding, Tim Finin, Anupam Joshi, Yun Peng, Rong Pan, and Pavan Reddivari, Search on the Semantic Web, TR-CS-05-09, 2005

32. Nenad Stojanovic and Ljiljana Stojanovic, Searching for the knowledge in the Semantic Web, American Association for Artificial Intelligence, 2002

33. FOAF Vocabluray, http://xmlns.com/foaf/0.1/

34. Intellidimension Semantic Web Search, http://www.semanticwebsearch.com/faq.rsp

35. Olivier Corby, Rose Dieng-Kuntz, and Catherine Faron-Zucker, Querying the Semantic Web with Corese Search Engine, Prestigious Applications of Intelligent Systems PAIS, ECAI, Valencia, 2004

36. U. Shah, T. Finin, and A. Joshi, Information retrieval on the Semantic Web. In Proceedings of the 11th international conference on Information and knowledge management, pages 461-468, 2002.

37. J. Mayfield and T. Finin, Information retrieval on the Semantic Web: Integrating inference and retrieval, InProceedings of the SIGIR 2003 Semantic Web Workshop, 2003.

38. T. Finin, J. Mayfield, C. Fink, A. Joshi, and R. S., Cost. Information retrieval and the semantic web, January 2004.

39. Zhijun Zhang and John A. Miller, Ontology Query Languages for the Semantic Web: A Performance Evaluation, Journal of Web Semantics (JWS), Vol., No. (2006)

40. Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl, RQL: A Declarative Query Language for RDF, WWW2002, May 7-11, 2002, Honolulu, Hawaii, USA. ACM 1-58113-449-5/02/0005

41. Eric Prud'hommeaux and Andy Seaborne, SPARQL Query Language for RDF, W3C Candidate Recommendation 6 April 2006, http://www.w3.org/TR/rdf-sparql-query

42. Koch, N. and Rossi, G., Patterns for adaptive web applications, In Proc. 7th European Conference on Pattern Languages of Programs, 2002

43. G. Adomavicius and A. Tuzhilin. "Personalization Technologies: A Process-Oriented Perspective." Communications of the ACM, vol. 48, no. 10, October 2005.

44. Erik Oskar Wallin , Individual Information Adaptation Based on Content Description, Thesis for the degree of Doctor of Technology - Royal Institute of Technology, Mars 2004

45. Matteo Baldoni, Cristina Baroglio, and Nicola Henze, Personalization for the Semantic Web

46. Koch, N. and Rossi, G. (2002). Patterns for adaptive web applications. In Proc. 7th European Conference on Pattern Languages of Programs.

47. Juan Danculovic, Gustavo Rossi, Daniel Schwabe and Leonardo Miaton, Patterns for Personalized Web Applications, EuroPLoP, 2001

48. Peter Dolog, Knowledge Representation and Reasoning in Personalized Web-Based e-Learning Applications, Znalosti 2006. Hradec Kralove. Czech Republic, February 2006

49. Flavius Frasincar and Geert-Jan Houben, Hypermedia Presentation Adaptation on the Semantic Web

50. Lloyd Rutledge, Jacco van Ossenbruggen, and Lynda Hardman, Making RDF Presentable — Global and Local Semantic Web Browsing, In: Proceedings of The Fourteenth International World Wide Web Conference (WWW2005) (pages 199-206), ACM Press, Chiba, Japan, May 2005.

51. Stefan Decker, Frank van Harmelen, Jeen Broekstra, Michael Erdmann, Dieter Fensel, Ian Horrocks, Michel Klein, and Sergey Melnik, The Semantic Web: the roles of XML and RDF, Internet Computing, IEEE, Sep/Oct 2000, Volume: 4, Issue: 5, On page(s): 63-73

52. Marcello Sarini and Carlo Strapparava, Building a User Model for a Museum Exploration and Information-Providing Adaptive System, Ninth ACM Conference on Hypertext and Hypermedia, Pittsburgh, USA, June 1998

53. Michael Denny, Ontology Building: A Survey of Editing Tools, November 2002, http://www.xml.com/pub/a/2002/11/06/ontologies.html

54. Michael Denny, Ontology Tools Survey, Revisited, July 14, 2004, http://www.xml.com/pub/a/2004/07/14/onto.html

55. Semantic Web Tools, http://esw.w3.org/topic/SemanticWebTools

56. Protégé , http://protege.stanford.edu/index.html

57. Protégé plug-ins, http://protege.stanford.edu/download/plugins.html

58. Using The Protege-OWL Reasoning API , http://protege.stanford.edu/plugins/owl/api/ReasonerAPIExamples.html

59. Ontoviz, http://protege.cim3.net/cgi-bin/wiki.pl?OntoViz

60. User Guide for Sesame, http://www.openrdf.com/doc/sesame/users/userguide.html

61. Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen, Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema, In The Semantic Web ISWC 2002

62. Natalya F. Noy and Deborah L. McGuinness, Ontology Development 101: A Guide to Creating Your First Ontology, Stanford University, Stanford, CA, 94305

63. McGuinness, D.L., Fikes, R., Rice, J. and Wilder S., An Environment for Merging and Testing Large Ontologies. Principles of Knowledge Representation and Reasoning:, Proceedings of the Seventh International Conference (KR2000).

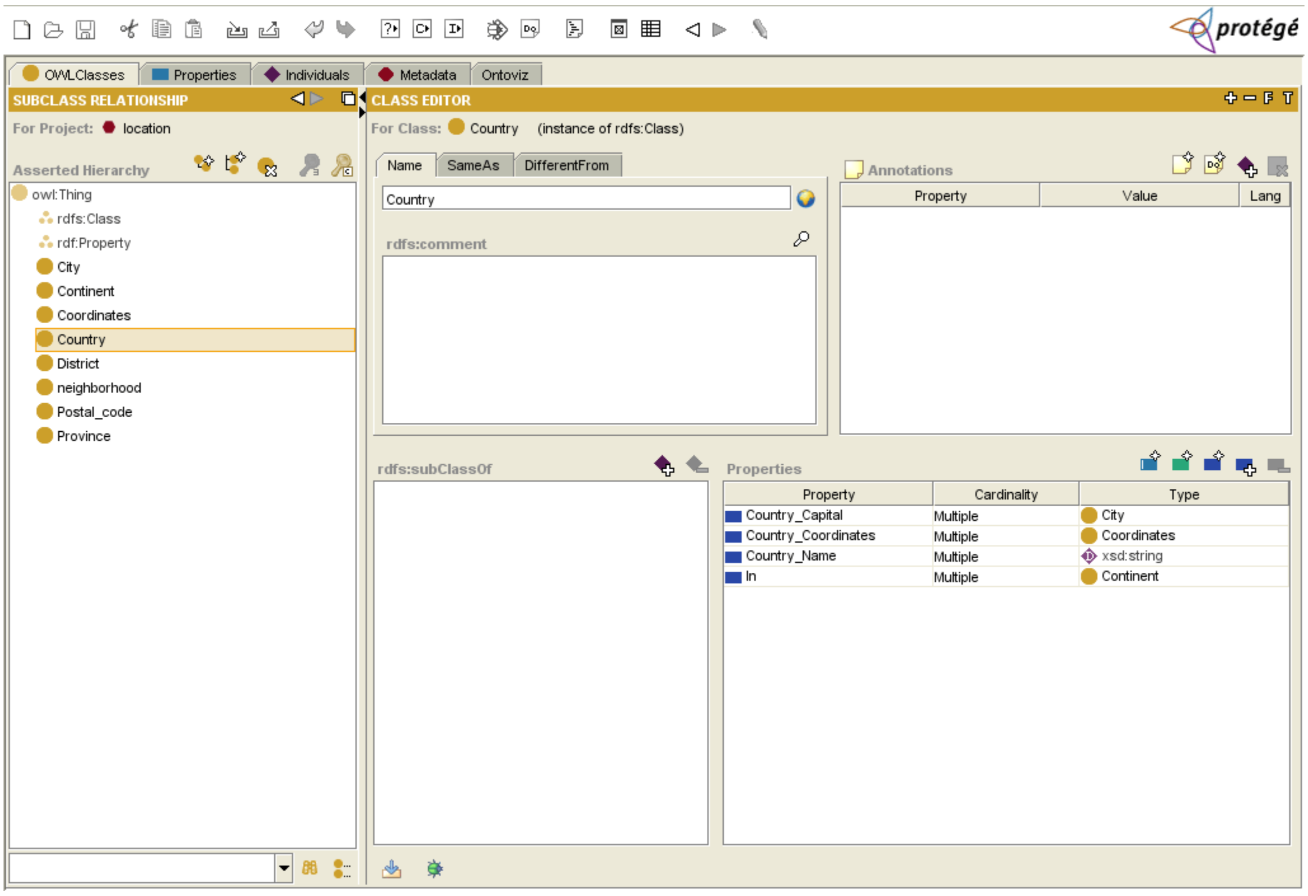
64. http://www.foaf-project.org/

65. http://xmlns.com/foaf/0.1/

66. Leigh Dodds, An Introduction to FOAF, XML.com http://www.xml.com/pub/a/2004/02/04/foaf.html, February 2004

Appendix

Location Ontology



For Project: location

For Class: Country (instance of rdfs:Class)

- owl:Thing
  - rdfs:Class
  - rdf:Property
  - City
  - Continent
  - Coordinates
  - Country
  - District
  - neighborhood
  - Postal_code
  - Province

Name: Country

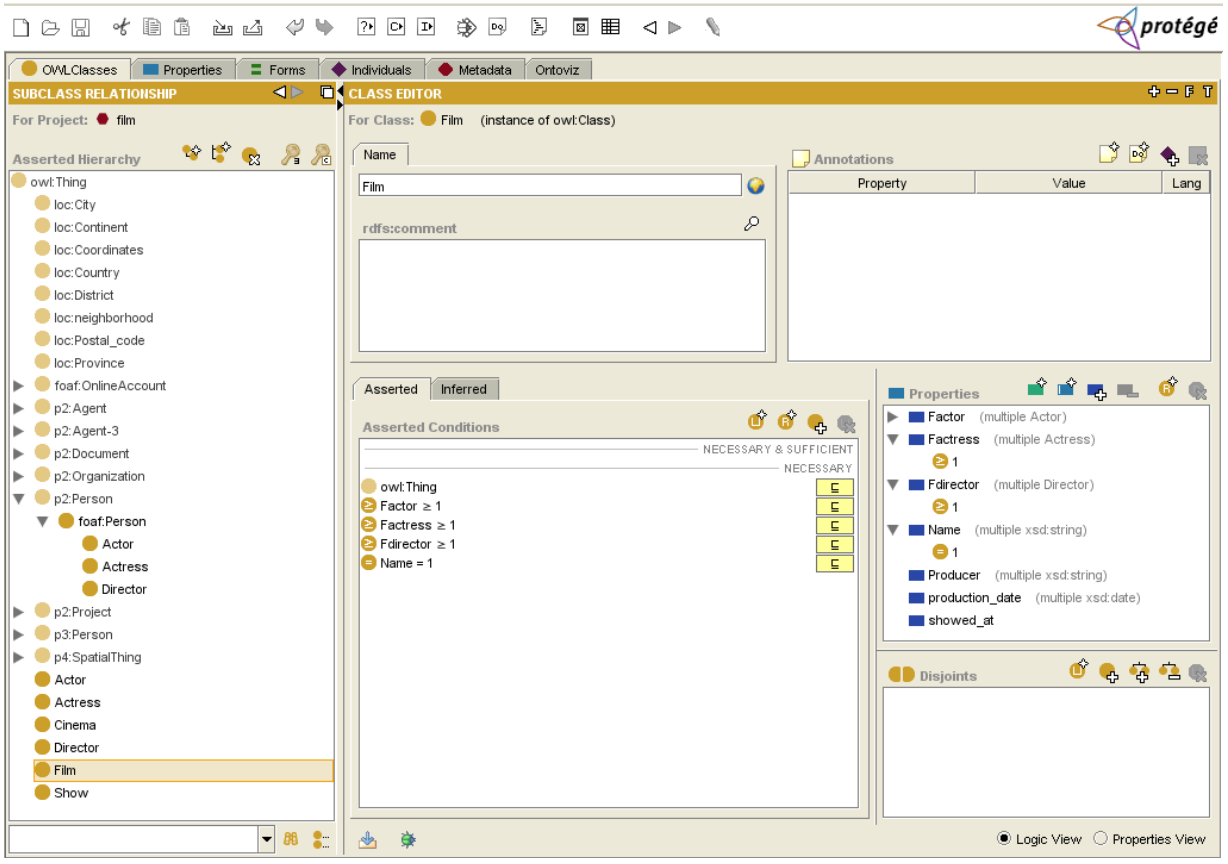
Property	Value	Lang

Property	Cardinality	Type
Country_Capital	Multiple	City
Country_Coordinates	Multiple	Coordinates
Country_Name	Multiple	xsd:string
In	Multiple	Continent

Figure 23: Location ontology in Protégé editor

Film ontology

Film ontology after importing the FOAF and location ontology



For Project: film

owl:Thing
- loc:City
- loc:Continent
- loc:Coordinates
- loc:Country
- loc:District
- loc:neighborhood
- loc:Postal_code
- loc:Province
- foaf:OnlineAccount
- p2:Agent
- p2:Agent-3
- p2:Document
- p2:Organization
- p2:Person
  - foaf:Person
    - Actor
    - Actress
    - Director
- p2:Project
- p3:Person
- p4:SpatialThing
- Actor
- Actress
- Cinema
- Director
- Film
- Show

For Class: Film (instance of owl:Class)

Name: Film

```
owl:Thing        ⊑
Factor ≥ 1       ⊑
Factress ≥ 1     ⊑
Fdirector ≥ 1    ⊑
Name = 1         ⊑
```

```
Factor  (multiple Actor)
Factress  (multiple Actress)
    ≥ 1
Fdirector  (multiple Director)
    ≥ 1
Name  (multiple xsd:string)
    = 1
Producer  (multiple xsd:string)
production_date  (multiple xsd:date)
showed_at
```

Figure 24: Film ontology in Protégé editor