CHAPTER 4

Semantic Web Technologies

- Introduction
- Web Services
- Semantic Web
- Semantic Web Service (SWS)
- Model for management of eLearning Resources
- N3Logic
- Reasoning Tools
- Conclusion
4.1 Introduction

The web is an impressive success story, in terms of both its available information and the growth rate of human users. The essential property of the World Wide Web is its universality. It is a large, disjointed reservoir of information composed of unsystematic but still interrelated repositories associated with particular domains and enterprises. Typical uses of the web today involve seeking and making use of information, searching for and getting in touch with other people, reviewing catalogs of online stores and ordering products by filling out forms etc.

The tools for finding information on Internet are search engines such as Google, Yahoo and others. But often users are not satisfied with the retrieved results that brings more noise than relevance. The main reason for this is the way in which information is represented on the web.

Most of the web’s content today is designed for humans to read, not for computer programs to manipulate meaningfully. Computers are only used as devices that post and render information; they do not have access to the actual content. The stored information in it is not semantically rich. If it were semantically enriched it would facilitate retrieval automatically or at least semi-automatically. Thus, computers can offer limited support in accessing and processing this information. To address this deficiency of the present Web, in the last few years the Web encountered two revolutionary concepts, one is enhanced Web Services and the other one is Semantic Web. But again Web service is syntactically rich, not semantically, whereas, Semantic Web is about semantics. In this chapter we discuss these two key technologies and the services emerging out of the two, i.e. Semantic Web services. We also discuss logic, specially the description logic and logic languages, such as, OWL, N3Logic. We also describe here some of the significant reasoning tools including the one we used in the present work.
4.2 Web Services

Web services changed the way applications communicate with each other on the web. They made it possible to integrate business operations within lesser time and cost of web application development and their maintenance, also promoted the reuse of code. Web services enabled businesses to share (or trade) functionality with arbitrary numbers of partners without having pre-negotiate communication mechanism or syntax representation [1].

Basically Web Service (WS) is a software component identified by an URI, which can be accessed via the Internet through its exposed interface. The interface description declares the operations, which can be performed by the service, the types of messages being exchanged during the interaction with the service, and the physical location of ports. A binding then defines the machine and ports where messages should be sent [2].

A Web Service can be defined as a network-accessible interface to an application. According to IBM Web Service tutorial, “Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes. ... Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service.”[3] The World Wide Web Consortium's (W3C) definition of a Web Service [4] as, “…a Web Service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via internet based protocols…”

4.2.1 Basic Web Service Architecture

The following architecture shows the common usage scenario of Web Services. The establishment of the set of standards for Web Services (e.g. SOAP, WSDL and UDDI) made it possible for developers to build Web
Services in the most common way. The model consists of 3 components, the service requester (SOAP Clients), which invokes services; the service provider (SOAP Server), which responds to requests; and the registry, where services can be published or advertised. The advertisement includes a service provider profile (e.g. company name and address); a service profile (e.g., name, category); and the URL of its interface definition (i.e., WSDL description).

![Figure: 4.1 Basic Web Service Architecture](image)

4.2.1.1 Simple Object Access Protocol (SOAP)

According to W3C, the SOAP (Version 1.2), “... is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics” [6].

In other words, it is a lightweight XML-based messaging protocol used to encode the information in Web service request and response messages before sending them over a network. SOAP messages are independent of any operating system or protocol and may be transported using a variety of Internet protocols, including SMTP, MIME, and HTTP and since it uses XML messaging over simple HTTP, it easily avoids firewall problems [7].
4.2.1.2 Web Service Description Language (WSDL)

WSDL is based on W3C XML standard and describes what Web services are, what they do, and how they can be accessed by applications that want to access them via SOAP. It is not really meant for humans to read. All standard Web Services are described in an associated WSDL document [8].

According to W3C, WSDL (version 1.1) is “…an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate” [9].

4.2.1.3 Universal Description, Discovery, and Integration (UDDI)

It is the first and foremost step for Universal Description, Discovery, and Integration, a platform-independent, XML-based registry for businesses worldwide to list themselves on the Internet. It is an open industry initiative, sponsored by OASIS, enabling businesses to publish service listings and discover each other and define how the services or software applications interact over the Internet [10].

The available services can be registered and published with a UDDI registry, which provides mechanism to be browsed and queried by other users, services and applications [1]. UDDI has two kinds of client, one is businesses that want to publish a service description and clients who want to obtain services of a certain kind and bind them through programs using SOAP [3].

4.2.2 Advantages of Web Services

The advantages of Web Services are as follows [11]:
4.2.3 Web Service Challenges
Though it has many advantages, still there are certain problems in Web Services, which we need to address.

1. Provided resources and services are not in machine understandable form, these are only in human understandable form.
2. The representation of resources and services on the Web are unstructured and they are loosely related to each other.
3. Searching resources and services on the Web at present is keyword based; no semantics of the resources are used. So by using some popular keywords, Web page owner can make his page visible. Often pages are lost and irrelevant search results are obtained because choice of keywords of endusers may often be different for the same page.
4. Very difficult to achieve Interoperability between services, tools etc.

4.3 Semantic Web
Semantic Web technology has drawn considerable attention of researchers in the field of distributed information systems, artificial intelligence, and so on. Researchers in the distributed information systems are taking interest to make use of Semantic Web technology as a central component of their software constructions.

The semantic web approach aims to develop languages for expressing information in a machine processable way. Tim Berners-Lee, who is the inventor of the World Wide Web (WWW), first envisioned semantic web
that provides automated information access based on machine-processable semantics of data. The explicit representation of the semantics of data, accompanied with domain theories (i.e. ontologies), will enable a web that provides a qualitatively new level of service [12].

4.3.1 Motivation for Semantic Web
The semantic web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. It is based on the idea of having data on the web defined and linked such that it can be used for more effective discovery, automation, integration, and reuse across various applications. [13]

Keyword-based search engines, such as Google, Yahoo and Alta Vista, etc., are the main tools for using today’s web. However there are serious problems associated with their use, like [14],

1. **High recall/ low precision:** Even if the main relevant pages are retrieved, they are of little use if thousands of other mildly relevant or irrelevant documents were also retrieved. Too much can easily become as bad as too little.

2. **Low or no recall:** Often it happens that we do not get any answer for our request, or that important and relevant pages are not retrieved. Although low recall is a less frequent problem with current search engines, it does occur.

3. **Results are highly sensitive to vocabulary:** Often initial keywords do not get the results we want; in these cases the relevant documents use different terminology from the original query. This is unsatisfactory because semantically similar queries should return similar results.

4. **Results are single web-pages:** If we need information that is spread over various documents, we must initiate several queries to collect the relevant documents, and then we must manually extract the partial information and put it together.
To realize the above mentioned Semantic Web vision the following are necessary [13]:

1. Automation- can computers do more (on the web)? The solution lies in making information on the web more “machine-friendly”.
2. Interoperability - combining information from multiple sources (so big organizations can avoid duplication)
3. Web services- discovery, composition. This would be a departure from the tool paradigm- instead of using computers like tools, make them work on our behalf. It also implies removing humans from the loop to the extent possible.

4.3.2 What is Semantic Web?
The word semantic implies “meaning” or, as WordNet defines it, “of or relating to study of meaning and changes of meaning.” In the term “semantic web”, ‘semantic’ indicates that the meaning of data on the web can be discovered- not just by people, but also by computers. It is a vision in which computers-software as well as people can find, read, understand, and use data over the World Wide Web to accomplish useful goals for users.

The semantic web is a vision of the next generation web, which enables web applications to automatically collect web documents from diverse sources, integrate and process information and interoperate with other applications in order to execute sophisticated tasks for humans [15].

4.3.3 Key Goals of Semantic Web
The key goals of Semantic Web are,

1. To build knowledge and understanding from raw data
2. Reuse of data
4.3.4 Semantic Web Components

Semantic web technology is built in a layered manner, i.e. it is processed in steps, each step built on top of another. The pragmatic justification of it is that it is easier to achieve consensus on small steps, whereas it is much harder to get everyone on board if too much is attempted [14, 16].

In building the semantic web in a layered manner, two principles should be followed:

1. **Downward Compatibility**: so agents (Agents are pieces of software that work autonomously and proactively [14]) fully aware of one layer should also be able to interpret and use information written at lower levels. E.g. agents aware of the semantics of OWL can take full advantage of information written in RDF and RDF Schema.

2. **Upward Partial Understanding**: agents fully aware of one layer should also be able to take at least partial advantage of information at higher levels. E.g. an agent aware of only RDF and RDF Schema semantics can interpret partial knowledge written in OWL, by disregarding those elements that go beyond RDF and RDF Schema.
Chapter 4: Semantic Web Technologies

The “layer cake” of the semantic web technology as shown in the above figure 4.2, describes the main layers of the semantic web design and vision.

4.3.4.1 HTML and XML

As we have seen in Figure 4.2, at the bottom of the Semantic Web layer is XML (eXtensible Markup Language) and XML Schema layer. XML is a subset of SGML (Standard Generalised Markup Language). Even today HTML (Hypertext Markup Language) is the most popular language. Web pages are written was developed from SGML because SGML was considered far too complex for Internet-related purposes. And XML was driven by shortcomings of HTML.

XML lets everyone create their own tags(-hidden labels) such as those that are used to annotate web pages of sections of text on a page. But it says nothing about what the structures mean. XML is particularly suitable for sending documents across the web.

4.3.4.1.1 Important Features of XML

Some of the important features of XML are [14],

1. Extensible: tags can be defined; can be extended to lots of different applications.
2. Markup language: which allow one to write some content and provide information about what role that content plays.
3. Allows the representation of information that is also machine-accessible.
4. XML document is, more easily accessible to machines because every piece of information is described. Moreover, their relations are also defined through the nesting structure. For example, the <author> tags appear within the <book> tags, so they describe properties of the particular book. A machine processing the XML document would be able to deduce that the author element refers to the enclosing books element, rather than having to infer this fact from proximity considerations, as in HTML.
5. Separates content from formatting: that means, the same information can be displayed in different ways, without requiring multiple copies of the same content; moreover, the content may be used for purposes other than display.

6. A meta-language for markup: it does not have a fixed set of tags but allow users to define tags of their own.

4.3.4.1.2 Issues with XML

XML is a universal meta-language for defining markup. It provides a uniform framework, and a set of tools like parsers, for interchange of data and metadata between applications. But it has also some limitations like,

1. XML does not ensure standard vocabulary and hence is open to interpretation. For example, one can use an element as ‘Author’, another can use it as ‘Writer’. Here, humans can make out that both are same, but how a machine/system decide. This creates confusion when machines try to share data with each other.

2. The nesting of tags does not have standard meaning; it is up to each application to interpret the nesting.

For example, David John is a lecturer of Thermodynamics. There are various ways of representing this sentence in XML. Two possibilities are:

```xml
<course name="Thermodynamics">
<lecturer>David John</lecturer>
</course>

<lecturer name="David John">
<teaches>Thermodynamics</teaches>
</lecturer>
```

The above two formalizations include essentially an opposite nesting although they represent the same information. In the first case, “course” name is considered as the primary one which nested the element “lecturer”. Whereas, in the second case, “lecturer” is treated as primary element and
nesting element is “teaches”, which is basically refers the course name. So there is no standard way of assigning meaning to tag nesting.

3. Domain-Specific Markup Languages: Since the user is at freedom to define his/her own tags, many domain-specific markup languages have been developed like, MathML [17] and CML (Chemical Markup Language) [18]. The problem with various domain-specific markup language is that of non-standardization while describing the resources on the Web.

But at the same time preventing this kind of flexibility and extensibility will again result in lack of inadequate resource description. Hence, there should be a common model/framework, which can bridge the gap between these various schemas. It is at this stage that the RDF came into the picture, which is also the next layer in the Semantic Web pyramid as shown in figure 4.2.

4.3.4.2 Resource Description Framework (RDF)

RDF is a basic data model, not a language. The RDF model provides the description of Web documents (in other words rendering of metadata to the documents) in a natural manner so that the metadata can be shared across different applications. RDF expresses the meaning, encoded in sets of triplets (resource/subject, predicate/property and object/value), each triplet being rather like the subject, verb and object of an elementary sentence. These triplets can be written using XML tags.

4.3.4.2.1 RDF Triplets
A simple RDF model has three parts [19],

1. Subject/Resource: Any entity, which has to be described, is known as resource, also known as subject. It can be a ‘webpage’ on Internet or a ‘person’ in a society.
2. **Predicate/Property**: Any characteristic of resource or its attribute, which is used for the description of the same, is known as property or predicate. For example, a webpage can be recognized by ‘Title’ or a man can be recognized by his ‘Name’. So both are attributes for recognition of resource ‘webpage’ and ‘person’ respectively.

3. **Object/Value**: A property must have a value also known as object. For example, the title of DRTC webpage is “Documentation Research and Training Centre”, name of a person is “S. R. Ranganathan”.

The combination of subject, predicate and object is said to be a ‘Statement’ or ‘Rule’. For example, a statement, David John is the author of the webpage http://drtc.isibang.ac.in/~David. This statement can be represented diagrammatically as follows:

![Figure 4.3: RDF statement](image)

The XML-based representation of the above statement is,

```xml
<?xml version="1.0" Encoding="UTF-16"?>
<rdf: RDF
 xmlns : rdf = "http://www.w3.org/1999//02/22-rdf- syntax-ns#"
 xmlns : mydomain = "http://mydomain.org/schema/"
<rdf:Description
 rdf:about = "http://drtc.isibang.ac.in/~David">
 <mydomain:author>David John</mydomain:author>
</rdf:Description>
</rdf :RDF>
```
The first line specifies that we are using XML. `xmlns:rdf = “http://www.w3.org/1999/02/22-rdf-syntax-ns#”` specifies the namespace. “An XML namespace is a collection of names, identified by a URI reference [RFC2396], which are used in XML documents as element types and attribute name. XML namespaces differ from the “namespaces” conventionally used in computing disciplines in that the XML version has internal structure and is not, mathematically speaking, a set” [20].

The syntax of declaring an XML namespace is,

```xml
xmlns:namespace-prefix = “namespace”
```

The `rdf:Description` element makes a statement about the resource `http://drtc.isibang.ac.in/~David`. Within the description the property is used as a tag, and the content is the value of the property.

The most important feature of RDF is that it is developed to be domain-independent, i.e. it is very general in nature and does not restrict/apply any constraint on any one particular domain. It can be used to describe information about any domain.

The RDF model imitates the class system of object-oriented programming. A collection of classes (as defined for a specific purpose or domain) is called a ‘schema’ in RDF. These classes are extensible through ‘subclass refinement’ [19]. Thus, various related schemas can be made using the base schema. RDF also supports metadata reuse by allowing transmission or sharing between various schemas.

4.3.4.2.2 RDF versus RDF Schema (RDFS) Layers
An illustration of different layers involved in RDF and RDFS [19] can be represented in the following way for a statement, *Networking is taught by David John*. 
The schema for this statement may contain classes such as lecturers, academic staff members, staff members, courses and properties such as is taught by, involves, etc. The above statement can be illustrated as follows. In the following figure 4.4, blocks are properties, ellipses above the dashed line are classes, and ellipses below the dashed line are instances.

4.3.4.2.3 Issues with RDF Schema

RDF and RDFS allow the representation of some ontological knowledge. The main modeling primitives of RDF/RDFS concern the organization of vocabularies in typed hierarchies: subclass and subproperty relationships, domain and range restrictions, and instances of classes. However, a number of other features are missing, like,

1. Local scope of properties: rdfs:range defines the range of a property, say ‘eats’, for all classes. Thus in RDF Schema we cannot declare range restrictions that apply to some classes only. For example, we cannot say that cows eat only plants, while other animals may eat meat, too.
2. *Disjointness of classes:* Sometimes we wish to say that classes are disjoint. For example, male and female are disjoint. But in RDF Schema we can only state subclass relationships, e.g., female is a subclass of person.

3. *Boolean combinations of classes:* Sometimes we wish to build new classes by combining other classes using union, intersection, and complement. For example, we may wish to define the class person to be the disjoint union of the classes male and female. RDF Schema does not allow such definitions.

4. *Cardinality restrictions:* Sometimes we wish to place restrictions on how many distinct values a property may or must take. For example, we would like to say that a person has exactly two parents, or that a course is taught by at least one lecturer. Again, such restrictions are not possible to express in RDF Schema.

5. *Special characteristics of properties:* Sometimes it is useful to say that a property is *transitive* (like “greater than”), *unique* (like “is mother of”), or the *inverse* of another property (like “eats” and “is eaten by”).

Thus we need an ontology language that is richer than RDF Schema, a language that offers the above features and more. In designing such a language one should be aware of the trade-off between expressive power and efficient reasoning support. Generally speaking, the richer the language is, the more inefficient the reasoning support becomes, often crossing the border of non-computability. Thus we need a compromise, a language that can be supported by reasonably efficient reasoners while being sufficiently expressive to express large classes of ontologies and knowledge.
4.3.4.3 Ontology
The concept originated more than two thousand years ago from philosophy and more specifically from Aristotle’s theory of categories [21]. The original purpose was to provide a categorization of all existing things in the world. Ontologies have been lately adopted in several other fields, such as Library and Information Science (LIS) (then not directly called 'ontologies', Artificial Intelligence (AI), and more recently in Computer Science (CS), as the main means for describing how classes of objects are correlated, or for categorizing what archivists generically call documents [22]. Many definitions of ontologies have been provided. According to Gruber ontology is defined as, “an explicit specification of a conceptualization” [23]. Later on Studer et al [24] extended the definition and defined ontology as "is a formal, explicit specification of a shared conceptualisation".

Studer’s definition includes the idea of shared in the notion of conceptualization and formal relations among the concepts. The explicit, formal representation of a shared conceptualization involves a perspective of a specific reality, and which is constituted in the conceptual structure of a knowledge base. Furthermore, the ultimate objective of ontology is to share the knowledge it represents. An ontology defines the terms and their formal relations within a given knowledge area.

We define ontology as, a set of shared conceptualizations with their formal relationships developed in a polynomial hierarchy.

Based upon the above discussions, we can draw the characteristics of ontology as follows,

1. **Shared** - the notion of ontology is to capture commonly agreed knowledge.

2. **Conceptualization** - refers to the mental formulation of a phenomenon in the world. It is developed by identifying the related concepts of that phenomenon. The shared conceptualization will
help in avoiding the lack of shared understanding between terms in one vocabulary as well as terms in various metadata vocabularies.

3. *Formal relationships* - ontology formalizes the relationships among the concepts, which makes computer to interpret the semantic relationships among the concepts and infer the implicit knowledge.

4. *Polynomial hierarchy* - it develops in a polynomial hierarchy order instead a rigid monolithic hierarchical structure.

4.3.4.3.1 **Ontology Benefits**

The benefits of ontology can be identified as follows [14],

1. Provide a shared understanding of domain
2. Useful in representing and facilitating the sharing of domain knowledge by human and automatic agents
3. Useful for the organization and navigation of web sites
4. Useful for improving the accuracy of web searches. Web searches can exploit the generalization and/ or specialization of information.

4.3.4.4 **Ontology Languages and Logic**

Semantic Web makes it possible to interact with logically connected data on the Web. Through a rich knowledge representation model, such as, RDF, Semantic Web provides a highly structured data. It is now possible for application developers in sharing their rich structured data on the Web and software agents can infer knowledge based upon the different kinds of data available on the Web. It is important to mention that RDF is built on the elementary pointer mechanism, Universal Resource Identifier (URI). We know in traditional Web, URI is mainly used to refer the documents and its parts through the hypertext mechanism. But the new emerging Semantic Web shows a new face of it by using it to name anything, starting from abstract concepts “color” to physical object “mountain” to electronic objects “home page of an institution”. RDF is also used to name the relationships between objects as well as the objects themselves [25].
In representing knowledge, logic plays an important role. Logics enhance the ontology language further. It helps to establish the consistency and correctness of data sets and to infer conclusions that are not explicitly stated but are required by or consistent with a known set of data. We list here some of the important features of logics as follows [14, 25],

1. Logic provides a high-level language in which knowledge can be expressed in a transparent way and will have a high expressive power.
2. It has a well-understood formal semantics, which assigns an unambiguous meaning to logical statements.
3. Automated reasoners can deduce (infer) conclusions from the given knowledge, thus making implicit knowledge explicit. For example,
   1. X is a Cat
   2. a Cat is a Mammal
   3. a Mammal gives birth to young ones

   Therefore, X gives birth to young ones

4. There exist proof systems for which semantic logical consequence coincides with syntactic derivation within the proof system.
5. Because of the existence of proof systems, it is possible to trace the proof that leads to a logical consequence. In this sense, the logic can provide explanations for answers.

However, addition of logic to the Web needs care as the Web with several characteristics, can lead us to the problems while we use the existing logics [25]. Addition of logic to the Web pre-supposes use rules to make inference, necessary courses of action, etc. It is important that the logic deployed must be powerful enough in describing the complex objects, but at the same time it must not be so complex and inflexible that it becomes contradictory for the software agents itself while inferring knowledge.
There are number of different knowledge representation paradigms that have emerged to provide languages for representing ontologies, in particular description logics (discussed in section 4.3.4.4.1) and frame logics. Web Ontology Language (OWL) is one such language based upon description logic in representing knowledge.

The other such languages belonging to the family of description logics are such as, Knowledge Interchange Format (KIF) [26], Simple Common Logic (SCL) [27] etc.

In the following sections description logic is discussed including OWL and its family members.

**4.3.4.4.1 Description Logics (DL)**

Description logics (DL) are closely related to First Order Logic (FOL) and Modal Logic (ML). Research on DL started to overcome computational problems of different complexity as the reasoning in different fragments of FOL. The research on DL started under the label “terminological systems” to emphasize that the representation language was used to establish the basic terminology adopted in the modeled domain [28] followed by “concept languages”. Now DL has become a cornerstone of Semantic Web for its use in designing ontologies.

DL became popular since the focus moved towards the properties of the underlying logical systems. Research on DL covered the theoretical foundation as well as the implementation of knowledge representation systems and the development of applications in several fields. For example, reasoning about database conceptual models; for schema representation in information integration system or for metadata management; as logical foundation of ontology languages, etc. [28].
Description logics are formal logics with well-defined semantics. Semantics of DL is defined through *model theoretic semantics*, which formulate the relationships between the language syntax and the models of a domain. In designing DL, the emphasis is given on key reasoning problem “decidability” and the provision of sound and complete reasoning algorithms. A key characteristic feature of DL is their ability to represent relationships beyond the “*is-a*” relationships that can hold between concepts [28].

In DL, the important notions of domain are described by “*concept descriptions*” that are built from *concepts* i.e., unary predicates and *roles* i.e., binary predicates by the use of various concept and role constructors. In addition, it is also possible to state facts about the domain in the form of axioms which act as constraints on the interpretations in a DL knowledge base [29].

In DL knowledge base the distinction between TBox and ABox is drawn which are the two main components of it. TBox contains intentional knowledge in the form of terminology and is built through declarations that describe general properties of concepts. In other words, it contains sentences describing concept hierarchies i.e. relation between concepts, while ABox contains extensional knowledge or assertional knowledge that is specific to the individuals of the domain of discourse [28].

4.3.4.4.2 *Web Ontology Language (OWL) and its Family Members*

The Web Ontology Working Group of W3C identified a number of characteristic use-cases for the semantic web that would require much more expressiveness than RDF and RDF Schema offer. The researchers in the United States (US) and in Europe identified the need for a more powerful language to build ontology. In Europe OIL (Ontology Interface Layer), an ontology language was developed. In US, DARPA (Defense Advanced Research Project Agency) had initiated a similar project called DAML
(Distributed Agent Markup Language). Latter on these two have been merged and came up with a single ontology language, DAML+OIL.

DAML+OIL in turn was taken as the starting point for the W3C Web Ontology Working Group in defining OWL, the language that is aimed to be the standardized and broadly accepted ontology language of the semantic web. Description logic is the logical foundation of OWL ontology language. OWL is built on top of RDF and RDF Schema. OWL adds more vocabulary for describing properties and classes. It also adds, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes [30].

The intent of OWL language is to provide additional machine-processable semantics for resources, that is, to make the machine representations of resources more closely resemble their intended real world counterparts [31]. In order to add the following listed capabilities to ontologies, OWL uses both URIs naming and the description framework for the Web provided by RDF [30]. The added advantages are,

1. Ability to be distributed across many systems
2. Scalability to Web needs
3. Compatibility with Web standards for accessibility and internationalization
4. Openness and extensibility

OWL ontology language consists of three sub-languages, such as, OWL Full, OWL DL and OWL Lite. These sub-languages differ by their power of expressiveness as discussed below. OWL species are discussed below along with their advantages and disadvantages.

**OWL Full:** It is the complete language and it uses all the OWL language primitives. It allows the combination of these primitives in arbitrary ways with RDF and RDF Schema.
Advantage: It is fully upward-compatible with RDF, both syntactically and semantically: any legal RDF document is also a legal OWL Full document and any valid RDF/RDF Schema conclusion is also a valid OWL Full conclusion.

Disadvantage: due to its greater expressiveness, it has become undecidable and therefore impractical for applications that require complete (or efficient) reasoning support. More expressive knowledge base leads to the complexity in terms of reasoning. Software agents will need more time (where time growth rate is exponential) to process a query.

OWL DL: supports the users who want the maximum expressiveness while retaining computational completeness i.e. all conclusions are guaranteed to be computable and is designed to regain computational decidability, i.e. all computations will finish in finite time. OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL corresponds to the SHOIN (D) description logic, a little less expressive language.

Advantage: It supports efficient reasoning.

Disadvantage: We lose full compatibility with RDF: an RDF document will in general have to be extended in some ways and restricted in others before it is a legal OWL DL document. Every legal OWL DL document is a legal RDF document.

OWL Lite: OWL Lite is OWL DL with more restrictions. It corresponds to the less expressive SHIF (D) descriptive logic. For example, OWL Lite excludes enumerated classes, disjointness statements, and arbitrary cardinality. The idea is to make it easy to start with and easy to implement
processors, so that people can begin using OWL Lite easily and later graduate to more complicated uses.

Advantage:
It is easier to grasp (for users) and easier to implement (for tool builders).

Disadvantage:
The expressiveness is more restricted.

The advantageous features of OWL DL and tools available for designing ontologies motivated us to use it in representing the knowledge base for the present work.

Example
This example shows an ontology that describes African wildlife.

Figure 4.5: Classes and subclasses of the African wildlife ontology

Table 4.1: OWL Ontology

```xml
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl = "http://www.w3.org/2002/07/owl#"
<owl:Ontology rdf:about="xml:base"/>

<owl:Class rdf:ID="animal">
```
Chapter 4: Semantic Web Technologies

Animals form a class.

Plants form a class disjoint from animals.

Trees are a type of plant.

Herbivores are exactly those animals that eat only plants or parts of plants.

Carnivores are exactly those animals that...
4.3.4.5 Trust Layer

At the top of the pyramid is the trust layer, which is the a high-level and crucial concept. The Web will achieve its full potential only when users have trust in its operation (security) and in the quality of information provided. The trust layer can emerge through the use of digital signatures and other kinds of knowledge, based on recommendations by trusted agents or on rating and certification agencies and customer bodies.

Each layer in the Semantic Web layer cake is seen as building on the layer below. Each layer is progressively more specialized and also tends to be more complex than the layers below it. The layers can be developed and made operational relatively independently.
4.4 Semantic Web Service (SWS)

Web Service technology proposed as a uniform technology, which allows uniform and universal access via web standards to software components residing on various platforms and written in different programming languages. These are well-defined, reusable, software components that perform specific, encapsulated tasks via standardised web oriented mechanism. But the major drawbacks of web service technologies are their inability in automatic discovery and composition. Human intervention and effort is required which make them unusable in the complex business environment. The current standard technologies for web services (e.g. WSDL) provide only syntactical description of their functionalities, without any definition about what the syntactic definition might mean.

The main problem is web services are lacking the semantic description, the semantic web researchers proposed to augment Web services with a semantic description of their functionality in order to facilitate their discovery and integration. This technology which is essentially a combination of Web Services with Semantic Web technology, is referred as Semantic Web Services (SWS) [32].

So, we can say that the SWS is an extension of web service with an explicit representation of meanings. SWS will support the automatic discovery, composition, and execution of Web Services. Hence, it has the potential to alter the way knowledge and business services are consumed and provided on the web.

Liliana Cabral, et al…[1], characterized the SWS infrastructure along three orthogonal dimensions, usage activities, architecture and service ontology. These dimensions relate to the requirements for SWS at business, physical and conceptual levels. Usage activities define the functional requirements, which a framework for SWS must support. The architecture of SWS defines the components needed related to the description of a Semantic Web
Service, and constitute the knowledge-level model of the information describing and supporting the usage of the service.

4.4.1 SWS Vision
The vision of semantic web Service is to offer flexibility in the whole development and execution cycle of a Web Service. The other motivating factors are,

*Automatic discovery:* an agent will be able to locate automatically an appropriate Web service. At present, for example, a student who wants to find a suitable course has to search on the Web using search engine or UDDI kind of registry, until he meets his requirement and finally manually he has to execute the service. By adding semantic markup into the Web Services [33, 34] the necessary information for the specification of the service is both in a computer interpretable and human-readable format. Therefore, an ontology enhanced search engine or an ontology enhanced service registry is able to do the same automatically and in most efficient way.

*Automatic invocation:* means the automatic execution of the service by a software agent without human involvement. If the execution is a multi-step process, then software agent needs to know how to interact with the service to complete the process. Ontologies, like OWL-S markup language, provide a computer-interpretable application interface (API) for the execution of these function calls. Any ontology-enhanced client application is able to understand the inputs, outputs, preconditions, and effects of the service calls. Hence, it is able to do the automatic execution of the service.

*Automatic composition and interoperation:* In most of the cases a single Web Service is not able to produce the desired result for an end user. Instead several Web Services have to be composed in the appropriate way in order to produce the required result. So a software agent has to be able to select and combine the number of Web Services to satisfy the user needs.
Here services also have to interoperate with each other to provide a valid solution.

**Automatic monitoring:** This is quite a complicated task. It mainly concerned with the monitoring during the execution of composite task, alerts about the unanticipated malfunction if any. The advantage of automatic monitoring is, it can give feedback to the execution or composition agents and according to that they will be able to adapt to the new situation, if any changes are required [35].

### 4.4.2 Architecture of SWS

Gugliotta, A., et al. [36] have discussed about the components of SWS from the architectural point of view. According to them SWS infrastructure contains the following components, underlining the security and trust mechanism. The components are:

1. a service register - provides the mechanisms for publishing and locating services in a semantic registry as well as functionalities for creating and editing service descriptions
2. a reasoner - used during all activities and provides the reasoning support for interpreting the semantic descriptions and queries
3. a discoverer and a matchmaker - mediate between the requester and the register respectively during the discovery and selection of services
4. a composer (or decomposer) – required for executing the composition model of composed services
5. an invoker - mediate between requester and provider or decomposer and provider
6. a mediator - manage the resolution of mismatching between request and the services

They have further mentioned that these components can have different names and a complexity of their own in different approaches.
4.4.3 Service Ontology for SWS

Another dimension of SWS is service ontology. The service ontology represents the capabilities of service itself and the restrictions applied to its use. It integrates at the knowledge-level the information which are defined using web service standards, such as UDDI, WSDL with related domain knowledge.

Service ontology generally includes the functional capabilities, such as input/output, pre-conditions, post-conditions, etc.; non-functional capabilities, such as category, cost, quality of service; provider related information, such as organization name, address, cost; task or goal related information of the service; quality and domain knowledge defining, such as the type of inputs of the service [35, 36, 41]. But the service ontology used for describing SWS relies on the expressiveness and inference power of the underlying ontology language supported by semantic web [36].

4.4.4 SWS Technologies

The languages for describing web services like, WSDL and their composition on the level of business processes (e.g. BPEL4WS) lack semantic expressiveness. There are many approaches or technologies have been driving the development of Semantic Web Service frameworks, like, First-order Logic For Semantic Web-Services (FLOWS), METEOR-S, WSDL-S, OWL Service Ontology (OWL-S) and Web Service Modeling Ontology (WSMO). OWL-S and WSMO are the most important and popular initiatives to describe Semantic Web Services.

4.4.4.1 METEOR-S

METEOR-S focuses on workflow management techniques for semantic web services. This project involves the creation and application of a broad variety ontologies related to data, function, non-functional/QoS and execution semantics to support the complete Web process lifecycle
Chapter 4: Semantic Web Technologies

109

encompassing annotation, discovery, composition, optimization and execution [37].

The main effort of this project is to add semantics to WSDL with the use of ontologies instead of developing a new language like OWL-S. This enhanced the descriptive power of widely accepted WSDL and UDDI. The extensibility feature of WSDL is used to add semantics to service descriptions and UDDI data structures, the so called tModels (technical model), are used to represent grouping of operation with their inputs and outputs [5].

The current direction in METEOR-S is towards creating a framework for autonomic web processes by using multi-paradigm reasoning, semantic modeling and operations research based optimization (using DL reasoning, ILP, MDP to create Web processes with self-CHOP - self-configuring, self-healing, self-optimizing and self-healing properties) [37].

4.4.4.2 FLOWS

The FLOWS model provides infrastructure for representing messages between services. The focus here is on the semantic content of a message, rather than, for example, the specifics of how that content is packaged into an XML-based message payload. It also provides constructs for modeling the internal processing of Web services. The goal of FLOWS is to enable reasoning about the semantics underlying Web (and other electronic) services, and how they interact with each other and with the "real world". FLOWS does not strive for a complete representation of web services, but rather for an abstract model that is faithful to the semantic aspects of service behavior [38].

4.4.4.3 WSDL-S

WSDL-S aims to provide a lightweight approach for creating semantic web service descriptions. Recent industry activity in Web Services standards has reinvigorated the business process integration community. By providing a standards-based framework for exchanging information dynamically on
demand between applications, Web Services show promise to address the process integration needs of enterprise application integration. However, WSDL, in its current form, suffers from the lack of semantics leaving the promise of automatic integration of applications written to Web services standards unfulfilled. WSDL-S provides simple extension to add semantics to represent the requirements and capabilities of Web services, which is essential for achieving automation in service discovery and execution. In fact, the need for semantics is pervasive in the complete lifecycle of Web services, encompassing description, discovery, composition, and choreography/orchestration, and involving a broad variety of semantics covering data, function/behavior, quality of service and execution [39].

4.4.4.4 OWL-S

OWL-S (formerly DAML-S) is the first well-researched Web Services Ontology and has numerous users from industry and academe. It provides the mechanism to describe the services offered by the service providers and the services needed by the service requesters.

OWL-S is defined as, “…OWL-based Web service ontology which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form. OWL-S markup of Web services facilitates the automation of Web service tasks, including automated Web service discovery, execution, composition and interoperation. Following the layered approach to markup language development, the current version of OWL-S builds on the Ontology Web Language (OWL)” [40].
Chapter 4: Semantic Web Technologies

Figure 4.6: Upper Ontology for Web Services

The above figure 4.6 provides three kinds of information/knowledge about the Web Services,

1. *What kind of service is provided by this Web Service?* – Provided by Service Profile. It is basically the advertisement about the capabilities of the services;
2. *How this Web Service works?* – Provided by Service Model; and
3. *How to use or access this Web Service?* – Provided by Service Grounding.

Here, 'presents', *is_described_by* and 'supports' are the properties of class Service. The classes Service Profile, Service Model and Service Grounding are the corresponding ranges of those properties.

The following paragraphs discuss about information Service Profile, Service Model and Service Grounding

*Service profile:* describes services for the purpose of discovery. The service descriptions and queries are constructed from the description of functional properties, such as IOPEs (i.e. inputs, outputs, preconditions and effects) and non-functional properties, such as serviceName, contactInformation and an unbounded list of parameters for defining additional information, like, an
estimation of the max response time, geographic availability of the service, concept type, quality of service. The profile class can be subclassed and specialized, thus it supports the creation of profile taxonomies, which subsequently describe different classes of services [41].

OWL-S Profile is also designed in modeling pre and post-conditions in addition to the types of input and output parameters of the Web services. However there is still no concrete formalism fixed for describing the conditions. This reduces constraint specification to the types of input and output parameters. So, it neither allows the specification of relationships between inputs and outputs nor constraints on the temporal structure of a Web service or preferences over functional and non-functional service properties [42].

Service/ process model: once service is discovered, there is no more use of service profile, as it does not provide any other information which can help in invocation. It is the service model that provides information for service invocation. In particular, a service model can be used by an agent in different ways that seeks service, like,

1. to validate the possible composition;
2. to perform depth analysis whether the service meets its needs;
3. to compose service descriptions by multiple, simple or atomic services in order to accomplish a specific task;
4. to control the activities of different participant of the invoked services during the enactment/ invocation of a service.

Here, the service can be treated as process. In OWL-S framework, there are three types of processes, atomic, simple and composite. A process can have arbitrary number of IOPEs. Though the Service Profile and Service/ Process model have different roles in the automatic service execution, they are two different representation of the same service and naturally they have to concise to each other [59]. That means, IOPEs of one to be reflected to
IOPEs of other. However, OWL-S framework does not explicitly dictate any such constraint. So, in case of inconsistency between Service profile and Service Model and still OWL-S expression to be validated, in such a case somebody would expect the interaction with the service to break at some point. So, it is evident that the definition of the IOPEs in the Service Profile should be made very carefully, so that, from one side no inconsistency occurs, and from the other side, the Service Profile is not overwhelmed or short of IOPEs [43].

Service grounding: describes how an agent can access a service. It defines the communication protocol, format of the exchange messages and other properties like, IP address of the machine hosting the service, port number, etc. In OWL-S, grounding deals with the concrete level of specification, where as service profile and service model are thought of abstract representations. Grounding can be thought of as a mapping from an abstract to a concrete specification of those service description elements that are required for the interaction with the service. The OWL-S framework uses the WSDL for grounding process [5].

Figure 4.7: OWL-S framework [44]
4.4.4.5 Web Service Modeling Ontology (WSMO)

WSMO is another upper layer ontology for describing various aspects of Semantic Web Services. It is the part of research project WSMF (Web Service Modeling Framework) that tries to improve the existing Web Services model and technology.

WSMO provides the ontological specification for the core elements of Semantic Web Services. WSMO is designed on the following principles, namely, Web Compliance, Ontology-Based, Goal-Driven, Strict Decoupling, Centrality of Mediation, Ontological Role Separation, Description vs. Implementation and Execution Semantics [45, 46].

WSMO (based on version 1.2) consists of four top-level elements for describing several aspects of Semantic Web Services. These are: Ontologies, Goals, Web Services, and Mediators.

The WSMO ontology elements are defined in a meta-meta-model language based on the Meta Object Facility (MOF) [46]. For complete item descriptions, every WSMO element is described by properties that contain relevant, non-functional aspects. These are based on Dublin Core Metadata Set and other service specific properties, like, Versioning Information, Quality of Service information (e.g. availability, stability), Owner and Financial.
It is worth to mention that if the requestor of the service and the Web Services that provides the service use the same ontology, the matching between the goal and the capability can be directly established. But in most of the cases they use different ontologies. This creates the communication problem between the requestor and the provider, which is nothing but the heterogeneity problem. To overcome it, WSMO introduced four types of Mediators. They are,

1. OO mediators link ontologies to ontologies,
2. WW mediators link web services to web services,
3. WG mediators link web services to goals, and finally,
4. GG mediators link goals to goals.

The idea of WSMO ontology is very similar to other upper layer ontology OWL-S. The main feature of WSMO is its simplicity, completeness, expressiveness over OWL-S and executability. However, at present OWL-S and WSMO are not enough mature enough for use [42].

4.4.5 Comparison Between OWL-S and WSMO

It is already mentioned that among all the initiatives towards Semantic Web Services, OWL-S and WSMO are the most important. It is also worth to
mention here that METEOR-S in comparison to WSMO, is highly technology oriented and it does not provide a conceptual model for the description of services and their related aspects. John Domingue, et al, mentioned [46] that, at the epistemological level the differences between OWL-S and WSMO are related to the principles of Ontological Separation and Centrality of Mediation. The differences between OWL-S and WSMO can be drawn as follows:

1. OWL-S is an ontology and a language to describe Web Services whereas WSMO is basically a conceptual model for the core elements (Ontologies, Web Services, Goals and Mediators) of Semantic Web Services.
2. In OWL-S, service profile expresses the existing capabilities (advertisements) and desired capabilities (requests); but WSMO separates capabilities (provider) and requester points of view (goals).
3. OWL-S provides default mapping to WSDL; WSMO also defines a mapping to WSDL but aims at an ontology-based grounding.
4. OWL-S does not have an explicit notion of mediator, mediation is a by-product of the orchestration process; where as WSMO regards mediators as key conceptual elements.
5. OWL-S is based on OWL (represent taxonomical knowledge)/SWRL (provide inference rule); where as WSMO is based on WSML, a family of languages with a common basis for compatibility and extensions in the direction of Description Logics and Logic Programming.

4.5 Model for Management of eLearning Resources
In this section we discuss about the resources used in modeling and constructing the ontologies, resources used in designing the learning perspectives and the tools used for reasoning over the learning knowledge base.
In order to build the ontologies, we used OWL-DL language and used Protégé [47], an ontology editor to build the ontologies. Though OWL, with underlying description logic has several advantageous features, it also has several shortcomings. Even though OWL is very expressive language, still it is less expressive for a Web like environment as it does not support quantified variables, rules, or a mechanism to distinguish 'which' document or person asserts 'what' [25]. In particular, description logic axioms are restricted to a tree-structure that disallows to model rule-like axioms commonly needed for complex data integration tasks that involve queries, views and transformations [42]. This led to several proposals for combining the ontology languages with rule-based languages. SWRL [48] is one such example of extension of OWL DL with rules. But as we already discussed in perspectives of description logics, the main goal in the definition of rule language for OWL is to keep decidability of the main reasoning problems. Unfortunately, SWRL is known to be undecidable. DL-safe [49], a fragment of SWRL has been proposed as a decidable rule extension [42]. In this perspective, N3Logic as proposed by Tim Berners-Lee found to be useful for our purpose is discussed in section 3.6.1. It is used in writing the rules as well as in querying the knowledge base. Also to infer the knowledge from knowledge base we used Euler, a backward-chaining reasoner as discussed in the following section 3.6.2 along with other important reasoning tools available.

4.6 N3Logic

N3Logic is a subset of First Order Logic (FOL), like Description Logics (DL), but is difficult to categorize as it differs from most traditional logics expressiveness. It allows rules to be expressed on the Web. N3Logic uses the Notation 3 (N3) syntax [50], a non-XML serialization of RDF models, extended to allow greater expressivity and extends RDF with a vocabulary of predicates. N3 aims in providing a common data model and a common syntax for logical information. This makes it possible to extend by defining new terms in the ontology.
N3Logic permits rules to be integrated with RDF. It provides certain built-in functions, which allow information to be used from the Web and reasoned over. The goal of N3Logic is to make a minimal extension to the RDF data model, which makes it possible to use the same language for rules and data. Since the rules and data can be represented using the same language, N3Logic provides simplicity and completeness.

Another important goal of N3Logic is as stated by Berners-Lee et al. in [25] that “…information such as ‘but not limited to rules’, which requires greater expressive power than the RDF graph, should be sharable in the same way as RDF can be shared. This means that a person should be able to express knowledge in N3Logic for a certain purpose, later independently someone else can reuse that knowledge for a different unforeseen purpose. As the context of the latter use is unknown, this prevents us from making implicit closed assumptions about the total set of knowledge in the system as a whole”. This makes N3Logic a monotonic nature, which is made so intentionally by the developers of it [25]. N3 does not allow to express NOT (negation) or OR (disjunction). By not allowing OR, it makes the reasoning steps simple. On the other side, by not allowing NOT, there is no way to state “negation as failure”. The fact is that world is much bigger than any one reasoner could tackle. The fact is if we do not know something, it does not mean that it does not exist. For example, if we do not know something, answering “does not exist” is false. This is the fact of closed world. In the open world i.e. in Web environment we may not know something and the answer would be “do not know” instead saying that “does not exist”. What we do not know is called open world reasoning.

N3Logic provides a set of predicates. Its vocabulary is a union of N3 syntax and a set of URI references as shown in the following table 4.2. These URI references are defined in the following namespaces, such as,

@prefix string: <http://www.w3.org/2000/10/swap/string#>
@prefix log: <http://www.w3.org/2000/10/swap/log#>
@prefix math: <http://www.w3.org/2000/10/swap/math#>
@prefix crypto: <http://www.w3.org/2000/10/swap/crypto#>
@prefix list: <http://www.w3.org/2000/10/swap/list#>
@prefix os: <http://www.w3.org/2000/10/swap/os#>
@prefix time: http://www.w3.org/2000/10/swap/time#

Table 4.2: Partial set of N3Logic predicates

<table>
<thead>
<tr>
<th>string:contains,</th>
<th>string:startsWith,</th>
<th>string:endsWith,</th>
</tr>
</thead>
<tbody>
<tr>
<td>string:matches,</td>
<td>string:greaterThan,</td>
<td>string:lessThan,</td>
</tr>
<tr>
<td>string:notGreaterThan,</td>
<td>string:scrape ...</td>
<td>log:truth,</td>
</tr>
<tr>
<td>log:implies,</td>
<td>log:edualTo,</td>
<td>log:conclusion,</td>
</tr>
<tr>
<td>log:includes,</td>
<td>log:semantics,</td>
<td>log:notIncludes,</td>
</tr>
<tr>
<td>log:supports,</td>
<td>... math:function,</td>
<td>math:list,</td>
</tr>
<tr>
<td>... math:difference,</td>
<td>math:edualTo,</td>
<td>math:greaterThan,</td>
</tr>
<tr>
<td>math:lessThan,</td>
<td>math:cos,</td>
<td>... crypto:verify,</td>
</tr>
<tr>
<td>crypto:md5,</td>
<td>crypto:sign,</td>
<td>... list:append,</td>
</tr>
<tr>
<td>... list:in,</td>
<td>list:last,</td>
<td>... os:argv,</td>
</tr>
<tr>
<td>... os:environ,</td>
<td>... time:day,</td>
<td>time:hour,</td>
</tr>
<tr>
<td>time:minute ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

N3Logic includes axiom schemas for each of these properties. The reasoners can use these axioms in evaluating formulas and bind variables. These properties are also called “built-in functions” and these can be used to provide a variety of functionalities as described in [25].

4.6.1 Relationship with RDF, RDFS and OWL

In this section we briefly describe the significant relationships of N3Logic with RDF, RDFS and OWL.

1. In describing the relationship with RDF, it is worth to mention that even though N3 syntax allows RDF expression, still it does not completely make use of RDF vocabulary. For example,

2. “rdf:type” of RDF is shortened to “a” and can be replaced with a direct use of the full URI symbol for rdf:type. There are many such instances are described in [51].

3. The expression of RDFS semantics in N3Logic is easy. A set of N3Logic rules for defining rdf:domain and rdf:range are as follows:
4. The shorthand notation, “equal to” (=) in N3Logic refers the same as “owl:sameAs” in OWL. This shorthand notation is used to state that subject and object are equal in a statement.

5. No attempt at connecting OWL DL language with the N3Logic.

6. Use of functional properties of a datatype conflicting with OWL DL. [51]

In conclusion we can assert that since N3Logic is based on N3 syntax, which provides human-readable syntax for RDF, it is easy to write and read. Not only easy to use but also N3 syntax provides quoting, variables and the implication operator. Another advantage with N3Logic is, it provides a set of built-in functions, which allows rules to access Web resources, define which inference can be drawn from specific Web documents, and other useful functionality such as mathematic, cryptographic and string.

4.7 Reasoning Tools
A reasoner (semantic) is a piece of software able to infer logical consequences from a set of asserted facts or axioms. The notion of a semantic reasoner generalizes an inference engine. The semantic reasoner provides a richer set of mechanisms in order to infer new facts from a set of asserted facts. In inferencing, there are four things are involved as follows,

1. Facts;
2. Rules;
3. Queries; and
4. Results.

In general, the inference rules are commonly specified by means of an ontology language. And often express using description language (DL) or similar to DL (e.g., N3Logic). There are also many reasoners, which use first-order predicate logic in order to perform reasoning.
The present reasoning engines can be grouped into two based upon the reasoning methodologies used in executing the inference rules. They are,

1. *Forward chaining*: starts with the available data and uses inference rules to extract more data (for example, user provided data) until a goal is reached. Following forward chaining mechanism, the inference engine searches the inference rules until it finds one where the antecedent (*If* clause) is known to be true. And once it is found, the reasoner can conclude or infer, the consequent (*Then* clause), and in result infer a new information (facts); and

2. *Backward chaining*: it is an inference method used in automated theorem proofs, proof assistants and other artificial intelligence applications. Backward chaining starts with a list of goals (or a hypothesis) and works backwards from the consequent to the antecedent to see if there is data available that will support any of these consequences. An inference engine using backward chaining would search the inference rules until it finds one, which has a consequent clause (*Then* clause) that matches a desired goal. If the antecedent (*If* clause) of that rule is not known to be true, then it is added to the list of goals. The backward chaining mechanism also sometimes called as “derivational”.

There is also another type of reasoning engines that use both the forward and backward chaining systems.

We already mentioned there are many reasoning tools are available based upon the above described mechanisms. For example, Jess, XSB, CWM, Euler, F-OWL, Pellet, Racer, Fact, Fact++, Hoolet etc. In this section we briefly describe some of the important tools for reasoning.
Jess (Java Expert System Shell): it is a small, light rule engine, originally inspired by the CLIPS expert system shell and the CLIPS (C Language Integrated Production System) language. Its powerful scripting language gives access to all of Java's APIs. Like CLIPS, Jess uses an enhanced version of the Rete algorithm (a very efficient mechanism for solving the difficult many-to-many matching problem) to process rules. Jess includes many unique features including backwards chaining, working memory queries, and Jess can also directly manipulate and reason about Java objects [52].

XSB: is a logic programming and a deductive database system. It extends Prolog with new semantic and operational features, mostly based on the use of Tabled Logic Programming or tabling. It allows for declaration of tabled predicates either automatically by the system or manually by the user, which can be used to program a number of applications in Non-Monotonic reasoning and Knowledge representation. Its default tabling strategy is called Local Evaluation which is efficient for returning all answers to a query, and is useful for application, namely program analysis [53].

Pellet: is a leading choice for systems where sound-and-complete OWL DL reasoning is essential. It fully supports the OWL DL specification. Pellet incorporates various optimization techniques, including novel optimizations for nominals, conjunctive query answering, and incremental reasoning. Pellet is based on the tableaux algorithms. Pellet supports a subset of SWRL, DL-Safe Rules. Pellet also supports most of the proposed features and profiles of OWL 2, a forthcoming revision of the OWL.

CVM (Closed World Machine): is a general-purpose data processor for the semantic web. It is a part of SWAP, a Semantic Web Application Platform. CWM is somewhat like sed, awk, etc. for text files or XSLT for XML. CWM is a forward chaining reasoner. It can be used for querying, checking, transforming and filtering information. Its core language is RDF, extended to include rules, and it uses RDF/XML or RDF/N3 serializations
as required. It can be noted that CWM was designed as a proof of concept for the standard Semantic Web layered architecture (as shown in figure 4.2). This fact implies that the implementation of CWM as a reasoner does not particularly focus on scalability or performance issues for large data sets or rule sets as in real-world applications [55].

**Euler**: is an inference engine supporting logic based proofs. It is a backward-chaining reasoner [56] enhanced with Euler path detection. The axioms are acquired from the Web and translated into a kind of logic program. The Euler proof engine uses the resolution inference mechanism and only follows Euler paths (the concept Euler found several hundred years ago) so that endless deductions are avoided. This means that no special attention has to be paid to recursions or to graph merging. Euler proof mechanism is a proof engine written in Java by Jos De Roo, that uses Euler paths to infer without fear of endless loops. It can parse Notation3, including N3 rules. It is also interoperable with W3C CWM via N3. Euler has implementations in Java, C#, Python, Javascript and Prolog. Euler provides a set of built-in functions in the following namespace.

e: <http://eulersharp.sourceforge.net/2003/03swap/log-rules#>

Some of the significant built-in functions of Euler are discussed in the following table 4.3.

<table>
<thead>
<tr>
<th>Table 4.3: Euler built-in functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>e:length- builtin that calculates the length of the subject list;</td>
</tr>
<tr>
<td>e:max- builtin to maximum of the subject list;</td>
</tr>
<tr>
<td>e:min- builtin to minimum of the subject list;</td>
</tr>
<tr>
<td>e:disjunction- similar to log:conjunction but is logical OR and is used for rules with disjunctions in the conclusion;</td>
</tr>
</tbody>
</table>
FaCT (Fast Classification of Terminologies)- is a Description Logic (DL) classifier that can also be used for model logic satisfiability testing. The FaCT system includes two reasoners, one for the logic \( SHF (ALC) \) augmented with transitive roles, functional roles and a role hierarchy) and the other for the logic \( SHIQ (SHF) \) augmented with inverse roles and qualified number restrictions), both of which use sound and complete tableaux algorithms. The important features of FaCT are [57],

1. Its expressive logic (in particular the \( SHIQ \) reasoner): \( SHIQ \) is sufficiently expressive to be used as a reasoner for the DLR logic, and hence to reason with database schemata;
2. Its support for reasoning with arbitrary knowledge bases (i.e., those containing general concept inclusion axioms);
3. Its optimized tableaux implementation (which has now become the standard for DL systems); and
4. Its CORBA (Common Object Request Broker Architecture) based client-server architecture.

Racer- is an OWL reasoner and inference server for the Semantic Web. The origins of Racer are within the area of description logics. The reasoning kernel in the middle is the core of the system and implements a highly optimized tableaux calculus for deciding the ABox consistency problem of the description logic \( SHIQ(D) \) (w.r.t. a background TBox). Ontologies or knowledge bases can be fed in various syntax and formats into RacerPro: for example RDF(S), OWL Lite, OWL DL, and a native Racer Syntax, etc. [58].
4.8 Conclusion
In this chapter we discussed the Semantic Web technologies. We started with web services followed by Semantic Web and Semantic Web Services languages and tools. We discussed about ontology, logic and logic languages, and the reasoning tools and techniques. We also discussed about the technologies and tools used in the present work in the section on model for management of eLearning resources. Next chapter presents the eLearning system developed using the above mentioned technology stack, with details of architecture and implementation.

4.9 References


17. MathML. http://www.w3.org/Math/


33. OWL-S 1.0 Release. http://www.daml.org/services/owl-s/1.0/


38. Semantic Web Services Ontology (SWSO), version 1.0. http://www.daml.org/services/swsf/1.0/swso/


40. OWL-S 1.1 Release. http://www.daml.org/services/owl-s/1.1/


53. XSB. http://xsb.sourceforge.net/index.html


55. CWM. http://www.w3.org/2000/10/swap/doc/cwm


57. The FaCT system. http://www.cs.man.ac.uk/~horrocks/FaCT/
